NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC QUALITY INSPECTED 2

THESIS

SYSTEM CONTROLLER HARDWARE AND EMBEDDED SOFTWARE FOR THE PETITE AMATEUR NAVY SATELLITE (PANSAT)

by

James Anthony Horning

September 1997

Thesis Advisor: Rudolf Panholzer
Co-Advisor: Randy L. Wight

19980109 098

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE System Controller Hardware and Embedded Software for the Petite Amateur Navy Satellite (PANSAT)	5. FUNDING NUMBERS
6. AUTHOR(S) Horning, James Anthony	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

   This thesis documents the design of the hardware and embedded software of a digital computer that provides autonomous control of the PANSAT spacecraft. The system was designed for use during a two year mission in a low earth orbit. The computer uses an Intel M80C186XL running at 7.3728 MHz, 512 kbytes of error-detection and correction RAM, 64 kbytes of ROM, and standard CMOS components to provide a general purpose microcomputer. The purpose of the computer is to control all subsystems of the spacecraft, perform analog-to-digital conversions, orchestrate duplicate hardware components to provide redundancy, and upload new software from a ground station. The hardware system was built on printed circuit boards which were manufactured by the Space System Academic Group and tested for proper operation. The embedded software was coded using 80186 Assembler and the C programming language, tested for proper operation, and placed into ROM as firmware.

14. SUBJECT TERMS PANSAT, Digital Computer, Embedded System, Device Drivers, Spacecraft Control	15. NUMBER OF PAGES 302
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

SYSTEM CONTROLLER HARDWARE AND EMBEDDED SOFTWARE FOR THE PETITE AMATEUR NAVY SATELLITE (PANSAT)

James Anthony Horning
Naval Postgraduate School
B.S.C.S., California Polytechnic - San Luis Obispo, 1989

Submitted in partial fulfillment
of the requirements for the degree of
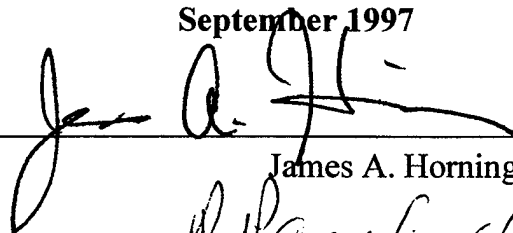
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
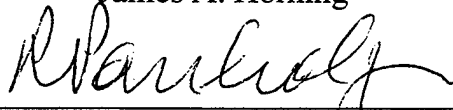
September 1997

Author: _____
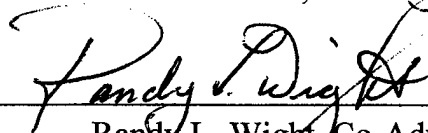
James A. Horning
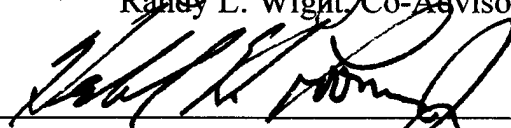
Approved by: _____

Rudolf Panholzer, Thesis Advisor

_____

Randy L. Wight, Co-Advisor

_____

Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis documents the design of the hardware and embedded software of a digital computer that provides autonomous control of the PANSAT spacecraft. The system was designed for use during a two year mission in a low earth orbit. The computer uses an Intel M80C186XL running at 7.3728 MHz, 512 kbytes of error-detection and correction RAM, 64 kbytes of ROM, and standard CMOS components to provide a general purpose microcomputer. The purpose of the computer is to control all subsystems of the spacecraft, perform analog-to-digital conversions, orchestrate duplicate hardware components to provide redundancy, and upload new software from a ground station. The hardware system was built on printed circuit boards which were manufactured by the Space System Academic Group and tested for proper operation. The embedded software was coded using 80186 Assembler and the C programming language, tested for proper operation, and placed into ROM as firmware.

TABLE OF CONTENTS

I.   INTRODUCTION

A.   Purpose

The purpose of this thesis is to document the design of a system that implements the digital computer of the Petite Amateur Navy Satellite (PANSAT). PANSAT is an experimental, low cost, lightweight, communications satellite that is currently being designed and built by officer students supported by the Space Systems Academic Group at the Naval Postgraduate School in Monterey, California. The design consists of the implementation details of the hardware as well as the low-level software that is embedded within the system. The computer, called the System Controller, incorporates error detection and correction memory for random-access memory, a read-only memory, specialized synchronous and asynchronous serial communications, analog-to-digital conversion, a parallel digital bus for subsystem control, and power detection and DC-DC conversion.

B.   Scope

Chapter II provides background information which includes a general description of the PANSAT project and its operational environment, radiation effects on electronic circuits, and general conventions used when writing this document. The third chapter presents an overview of the organization of the electronics of PANSAT, and a description of the hardware and software architectures of the System Controller. Chapter IV discusses the hardware design in detail. The fifth chapter describes the software device drivers design in detail and Chapter VI examines the higher-level software routines which use the device drivers. Chapter VII examines the testing of the hardware and software, presents recommendations, and ends with the conclusion. Several appendices follow containing the hardware schematics, the bill of materials for the hardware, software block and flow diagrams, software source code, and software generation facilities.

2

II. BACKGROUND INFORMATION

A. THE PANSAT PROJECT

The PANSAT project began in 1989 as an educational program for students at the Naval Postgraduate School's (NPS) Space Systems Academic Group (SSAG). The project goal is to provide meaningful and realistic research topics for students in the area of space systems engineering and space systems operations. In doing so, the Space Systems Academic Group has prepared students for space related tasks and has developed an infrastructure of facilities and personnel capable of developing space qualified systems.

PANSAT is a small satellite for digital store-and-forward communications in the amateur frequency band. It features a direct sequence spread spectrum differentially coded, binary phase shift keyed (BPSK) communication system at an operating frequency of 436.5 MHz. The store-and-forward capability will allow NPS and amateur radio operators to send or receive messages during several short communication windows every day, each 4 to 8 minutes in duration.

The entire satellite structure weighs approximately 150 pounds, has a diameter of about 19 inches, and is designed to be launched as a secondary payload from the Space Shuttle via the Hitchhiker Program. PANSAT is a 26-sided polyhedron, as shown in Figure 1, a configuration chosen to maximize solar panel area and thus power generation. PANSAT is not stabilized and will tumble freely once put into space. The satellite uses an omni-directional antenna system consisting of four quarter wave-length segments to achieve near uniform signal coverage regardless of PANSAT's orientation while tumbling in space.

PANSAT requirements specify a two-year mission life in a low-earth orbit (LEO) with an inclination between 28.5° and 90.0° [Ref. 1]. The Space Systems Academic Group has signed a Memorandum of Agreement with NASA Space Test Flight programming, ensuring a future flight onboard the Space Shuttle. Space Shuttle operational limits are altitudes between 203.7 km and 611.2 km and inclinations between 28.5° and 57.0° [Ref. 2].
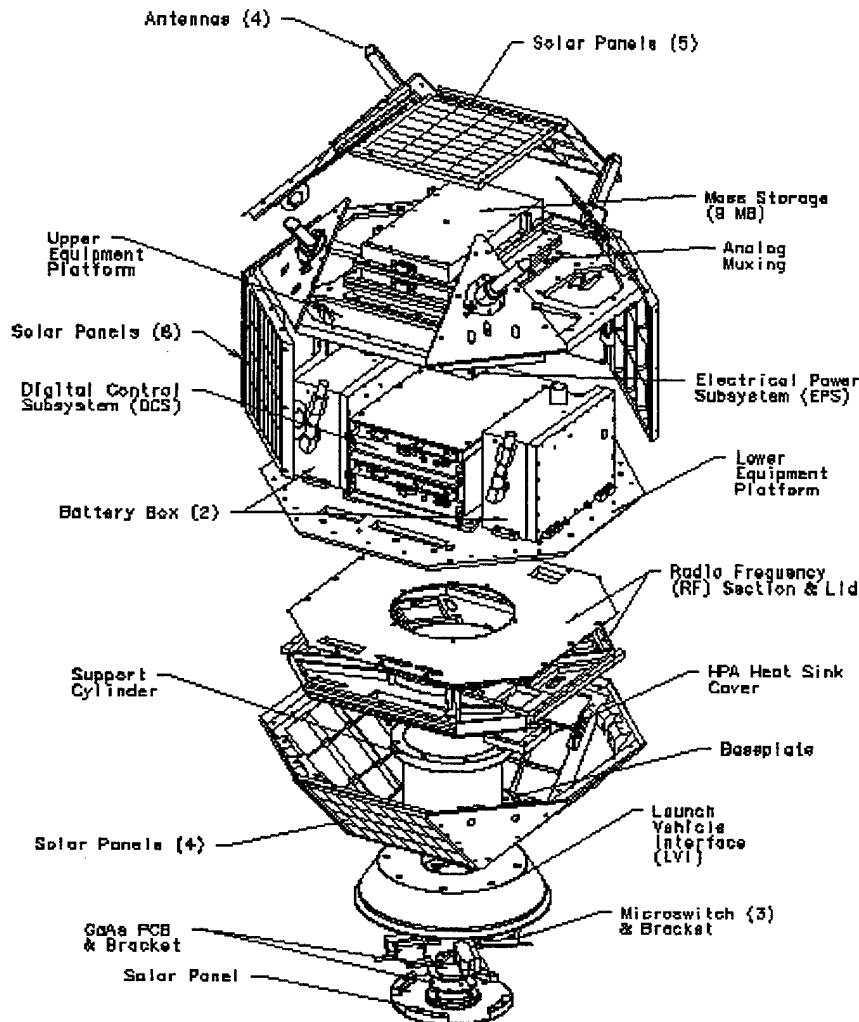
Figure 1. PANSAT Structure.

B. MISSION LIFE AND OPERATING ENVIRONMENT

1. Thermal Environment

Thermal analysis for temperatures of electronics inside PANSAT examines two situations: a hot case and a cold case, depending on solar flux and Sun orientation, the Earth, and internal power dissipation within the satellite. The cold case expects the temperature inside during operation to be between -15°C and -6°C, and the hot case predicts temperatures from about -4°C to 13°C [Ref. 3]. While these temperatures are well suited for the operation of integrated circuits (IC), the batteries prefer a warmer environment; this will be addressed briefly in the section describing the Battery Charge Monitor, in Chapter VI.

2. Operational Environment

PANSAT electronics are expected to provide continuous operation throughout the life of the satellite. Solar panels will provide sufficient energy during the sun-soak portion of an orbit to allow continuous operations of the satellite and to store energy into the batteries for the eclipsed portion of the orbit. The PANSAT System Controller design incorporates two redundant, mutually exclusive, system controllers which are prevented from simultaneous operation because of the switching design within the electrical power system (EPS).

3. Radiation Environment

In LEO, PANSAT will be significantly protected from cosmic radiation and the solar wind due to the shielding effects of the Earth's magnetic field [Ref. 4, p. 662], as well as the thickness of the structure itself. The primary source of radiation in LEO is low energy electrons, and low energy protons which are encountered mostly in the South Atlantic Anomaly (SAA) from approximately 45° latitude and 45° longitude centered near 20° N, 20° W [Ref. 5, p. 2339; Ref. 6, p. 2344]. Energetic protons, an occasional source of radiation, are expected with solar flares [Ref. 4, p. 712]. As a result PANSAT should experience an ionizing radiation dose rate of about one krad (Si) per year [Ref. 4, p. 452].

C. RADIATION EFFECTS ON ELECTRONICS

There are two primary effects caused by radiation on electronics: an effect from the dose rate that causes single event upsets (SEU), as well as a total dose effect. PANSAT will orbit in a relatively benign radiation environment compared to other regions of space. The selection of PANSAT electronics incorporate some concern for radiation exposure; however, the structure of the spacecraft including the boxes which contain the electronic modules provides substantial radiation shielding. As a lowest common denominator of electronic component selection where redundancy applies, industrial temperature grade integrated circuits fabricated with Complementary Metal Oxide Semiconductor (CMOS) technology using epitaxial layers are used. Circuitry of PANSAT that presents a source of a single point of failure uses radiation hardened devices. Otherwise, these radiation hardened devices are expensive, power-hungry, and an unnecessary choice for PANSAT.

1. Single Event Effects

Single event effects (SEE) are the responses of an IC to the passage of a single highly energetic charged particle, and include single event upsets (SEU), single event latchup (SEL), and single event burnout (SEB). As a particle travels through the silicon layers of an IC, it loses energy creating ionization or electron-hole pair generation along its path. Generally, the higher the mass and charge of the particle, the greater the amount of ionization produced.

SEBs are observed in power MOSFETs where high voltages and electric fields are present; and thus are not a real concern for PANSAT electronics of the System Controller. However SELs, the effect of activating a parasitic silicon controlled rectifier within a CMOS IC, has the potential to destroy a device or a portion of a device while it is in the latched condition. In order to reset the device it must be powered off. Fortunately, thoughtful device design can reduce or eliminate this effect. The SEU is the most common effect to upset an IC, resulting in a temporary or permanent change of state. SEUs are the result of the rate of individual hits on electronics, in particular an IC by high energy particles. SEUs result in errors within electronic systems by causing a change in the state of a logic storage element [Ref. 7].

2.   Single Event Effects Experienced by PANSAT

PANSAT is expected to experience a variety of SEEs throughout its lifetime. At worst, these effects will most likely cause a particular system to reset; initiating a restart of the electronics, as if a launch has just occurred. The design of PANSAT allows the electronic systems to be powered down, and then back up. Such action clears such SELs. Fortunately, PANSAT is not expected to experience many SELs.

D.   DOCUMENTATION CONVENTIONS

1.   Numbering

By default, all numbers in this thesis are base-10 (decimal). Base-16 numbers (hexadecimal) are prefixed by the following notation, *0x*. Thus, the number 0x0F is the hexadecimal value for 15 (decimal); an exception to this rule is found only within the assembly language excerpts of the software module *startup.asm* where the convention of the assembler is to append an *h*, e.g. 0Fh. Binary values are either evident (only one digit exists), or are pointed out within the text.

2.   Signal Names

Signal names are chosen to best represent the function they perform. In addition, digital signals also include the logic assertion level. If a signal has an overbar across its name, e.g. $\overline{RD}$, then that signal is considered active LOW; that is, it is considered asserted when the signal is a binary 0, corresponding to 0 V. Otherwise, the signal is considered active HIGH; that is, it is considered asserted when the signal is a binary 1, corresponding to 5 V.

3.   Logic Expressions

Logic equations for circuit diagrams and Karnaugh map analyses presented in this document use the + symbol for the inclusive logical OR function. The logical AND function is represented with the symbol *. The overbar either indicates the assertion level of a signal (as explained in the section above)

6

when it is the result of an equation, or indicates the negation of a logic expression. Otherwise, for programming languages (i.e. C), the operator rules of the language apply [Ref. 8, Ref. 9].

4. Software Flow Diagrams

Software flow diagrams use blocks to depict either a subroutine or a statement. Bold blocks refer to subroutines.

The project background and expected operating environment were considered in designing the PANSAT System Controller hardware and software. In addition, the System Controller design was influenced by the other electronic modules of the spacecraft. An overview of the hardware and software designs as well as an introduction to the spacecraft electronic modules are presented in the next chapter.

III. PANSAT ELECTRONICS AND SOFTWARE OVERVIEW

This chapter provides design constraints and tradeoffs that influenced the design of the System Controller. Furthermore, an overview of the electronic systems of PANSAT is presented as well as an architectural overview of the System Controller hardware and software.

A. DESIGN CONSTRAINTS AND TRADEOFFS

The design objective was to build a digital computer (system controller) using readily available components. Six major constraints influenced the design:

- Suitability for use in a short duration low-Earth orbit radiation environment.
- Printed circuit board area required.
- Speed of operation.
- Power required.
- Cost.
- High-level software environment support and compatibility with existing software tools.

PANSAT will operate on a limited power budget; all power is dependent upon solar panels with batteries to store power. Generally, faster systems require more power. Furthermore, radiation hardened components usually require more power than the non-hardened counterparts. Also, radiation hardened parts are usually extremely expensive (averaging 20 times the cost of a comparable high reliability, non radiation hardened part). The cost of using radiation hardened parts within PANSAT is neither justified nor necessary.

In general CMOS was chosen over other logic families because of its lower power consumption. Often, standard small scale integration (SSI) and medium scale integration (MSI) parts were chosen for logic generation because they are readily available and offer decreased susceptibility to SEUs at a reasonable cost. Since PANSAT System Controller operates at a relatively low clock rate, power was judged as less critical than size and functionality.

Two families of CMOS logic ICs are used throughout the design of the System Controller, High-speed CMOS and Advanced CMOS [Ref. 10]. The Advanced CMOS (AC) is approximately three to five times faster than the equivalent High-speed (HS) devices; however, the AC devices consume about 50% more power. AC components are also more expensive. AC components are used in the design only where necessary. They are mostly found in the memory glue logic.

For many LSI and VLSI devices in this design, high reliability (not radiation hardened), military temperature rated components are used wherever possible. Many devices are processed to the MIL-STD-883 specifications. A few others are processed to industrial specifications. All circuit components are identified in Appendix C which contains the Bill of Materials (BOM) for the circuit board fabrication.

B. PANSAT SUBSYSTEM HARDWARE GENERAL DESCRIPTION

Figure 2 shows a block diagram of the entire PANSAT electronics. Redundant modules: System Controller (SC), Analog Temperature Multiplexers (TMUX), and Mass Storage (MS), are shown above and below the common electrical bus called the Peripheral Control Bus (PCB). Primary and redundant module designation is arbitrary since respective modules are essentially identical. Redundant modules in standby mode will be powered off. However, because the control bus allows individual module addressing by the active SC, either, or both, of the modules (TMUX or MS) may be enabled. The two remaining modules of the PANSAT electronics are the Radio Frequency (RF) system responsible for up and down conversion of the communication signals, and the Electrical Power Subsystem (EPS) which is responsible for the distribution of power on the satellite.
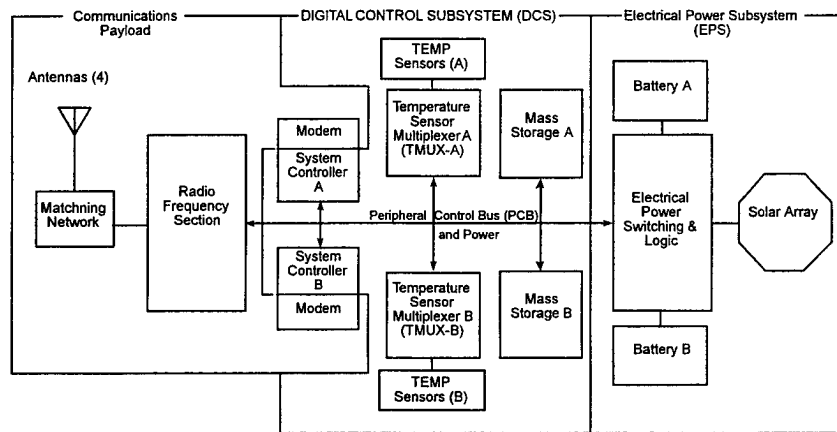
Figure 2. PANSAT Block Diagram.

The EPS controls the charging of the batteries via the solar panels and the distribution of power to the rest of the spacecraft. The satellite has 18 solar panels for the production of power. Power is stored in two banks of nine batteries each. Each battery bank has the capability of providing complete power for the satellite. The mass storage system provides the memory needed for storage of telemetry and data uploaded to the satellite by users of the satellite. It contains two redundant systems, each with 4.5 Mbytes of random access memory. The TMUX is an analog multiplexer which directs which temperature sensor is being read by a System Controller. There are temperature sensors on all major satellite components to provide a monitoring capability.

C. SYSTEM CONTROLLER ORGANIZATIONAL OVERVIEW

The System Controller of PANSAT is an embedded microprocessor-based computer system for the satellite. The electronics provide interface circuits to control all of the satellite's subsystems, as well as create an environment capable of supporting high-level software. Firmware, embedded within ROM in the computer, is the software that is capable of initializing the entire satellite, maintaining the batteries, and conducting simple communications with Earth with the goal of uploading more sophisticated software.

11

1. Hardware Organization

As mentioned above, the PCB is an 8-bit parallel control and data bus capable of uniquely addressing each subsystem in the spacecraft. Each subsystem has a unique power connection, switchable by command of the SC via the EPS. Also, each subsystem has a unique address in which it responds to commands issued by the active SC to perform reads and writes between the SC and the subsystem. Each subsystem has radiation hardened circuitry to interface the subsystem to the PCB. These components were chosen because this bus is not redundant and is a single point of failure. The electronic circuits of the PCB are always powered on when the spacecraft has bus power. These electronic circuits are responsible for isolating a subsystem from the bus when it is not powered on; the same circuits enable each subsystem to respond when powered on and addressed via a SC.

The System Controllers differ from the other subsystems in that they are capable of controlling the PCB (rather than responding to the PCB). However, within the EPS is circuitry which allows only one SC to be powered on at a time in order to remove the possibility of two controllers manipulating simultaneously the PCB. A SC remains powered on unless it fails to notify the EPS within a certain time interval; thus a SC is subject to an external watchdog timer contained within the EPS.

The system controllers are identical in design. However, the remaining discussions will normally refer to a single system controller, implying the same applies to the other SC. A SC is best understood by viewing its design from a top-down approach. As seen in Figure 3, at the top-most layer is a M80C186XL-10 microprocessor and some of its support circuitry for a clock and reset circuitry, to the right side are four other blocks. On the top-right is the general control input/output (I/O) module, the next block down on the right is the serial communications control module, next is the system memory module, and at the bottom is the Analog-to-Digital module. Also shown is the signal interface between the modules, required within a System Controller. A photo of the completed circuit board is shown following the block diagram.
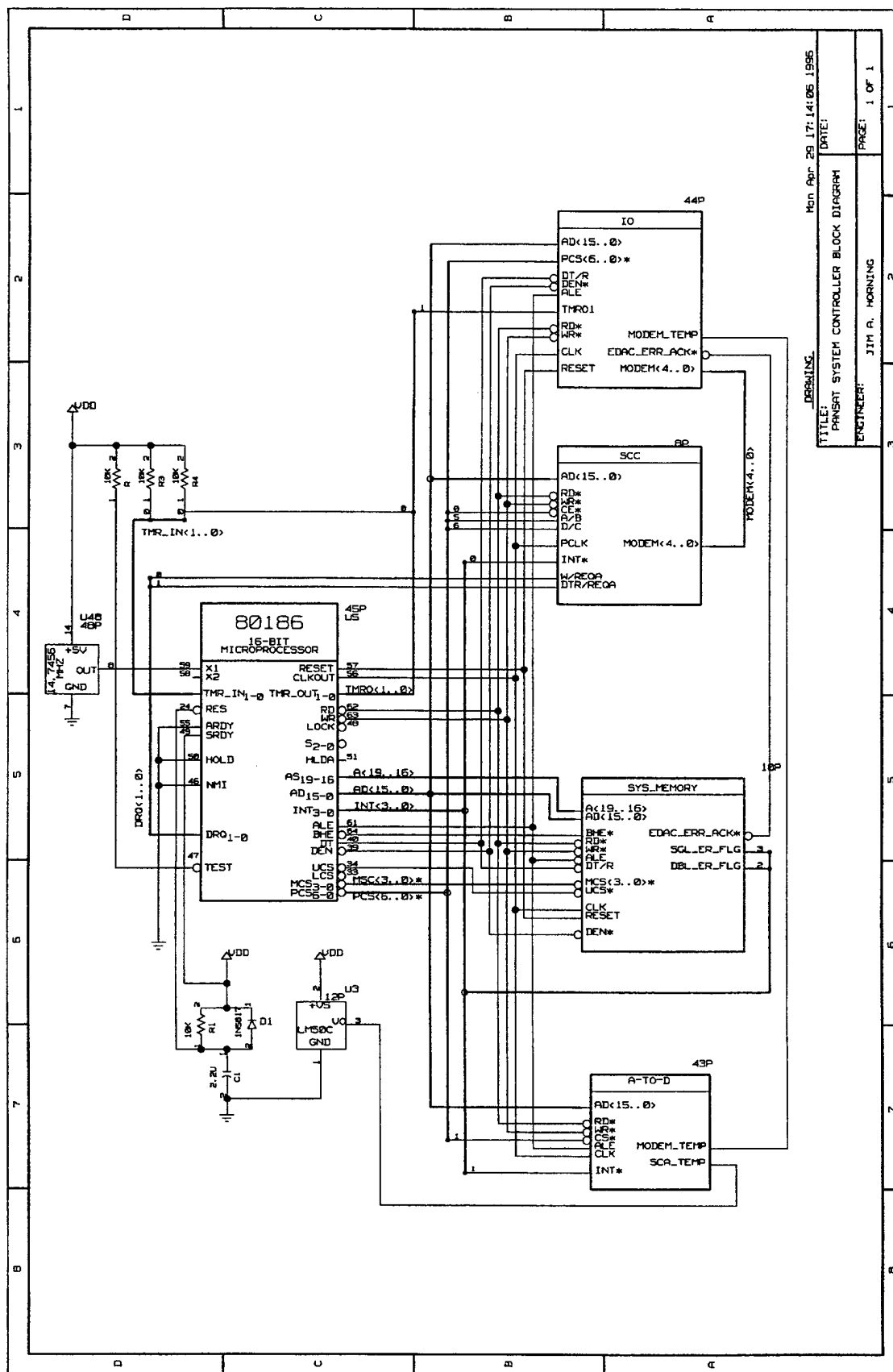
Figure 3. System Controller Block Diagram.

13
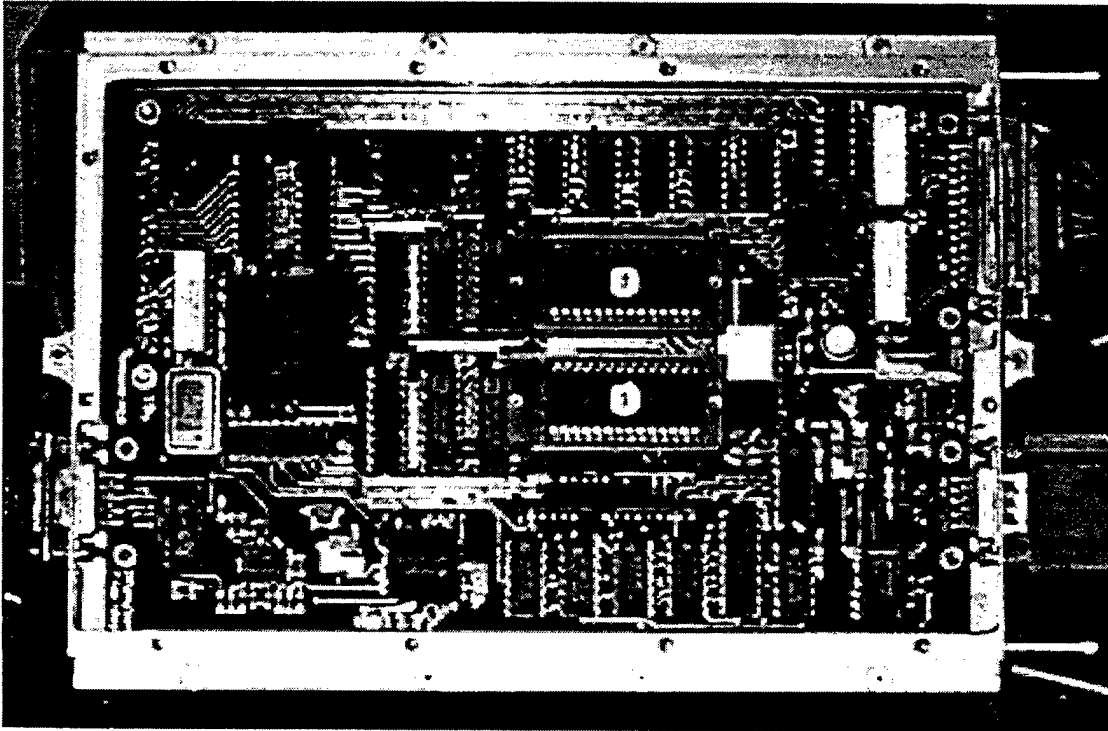
Figure 4. System Controller Circuit Board.

The M80C186 is available in many versions, each with a different maximum input clock rate. Within this document, *80186* will refer to any version of the microprocessor, and *M80C186XL* will refer specifically to the XL-10 (10 MHz) version which is used on board PANSAT.

2. Software Organization

The software organization of a SC resembles closely the hardware. However, since software not only provides control of hardware but also involves a process flow (the goal of the software is to do something from *start* to *finish*), further modules are useful in implementing the software for PANSAT.

The goal of the software described within this document is to provide simple and reliable control of the spacecraft in order to upload high-level layers of software which are modifiable from a ground station. This software is known as the ROM Boot software. It is embedded on space qualified ROM, and is thus a permanent product, also known as firmware.

Looking at the ROM boot software as in Figure 5, the first software module is responsible for the System Controller Startup. Next is the Boot Loader software which works as a large loop, making sure all the other modules orchestrate together correctly. Within this boot loading process, there are modules

responsible for the control of the batteries, responding to communications requests from a ground station, and dealing with scenarios which require alternate configurations of hardware.

PANSAT ROM Boot.

```
 _____                    _____
/Microprocessor/ ------------> |  Startup. |
/  RESET.      /                |_____|
/_____/                      |
                                     v
                               _____
                              | Boot Loader.|
                              |_____|
                                     |
                                     v
                               _____
                              | Secondary |
                              |  Loader.  |
                              |_____|
                                     |
                                     v
                               _____
                              | Final Loader.|
                              |_____|
```
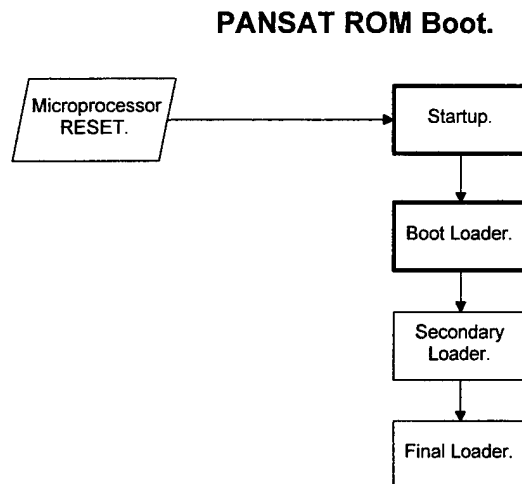
Figure 5. ROM Boot Software Overview.

Viewing from the software module perspective, Figure 6, there are device drivers for the CPU setup, the EDAC RAM, the PCB, the mass storage units, the serial communications, and telemetry gathering which includes driving the analog-to-digital circuits. This figure depicts all of the files required to describe the code of the entire ROM Boot software.

This overview presented high-level descriptions of the System Controller hardware and software in order to discuss in more detail the designs of these systems. The next few chapters describe in great detail the hardware and software designs of the System Controller, beginning with a presentation of the hardware that implements the satellite's general purpose computer.
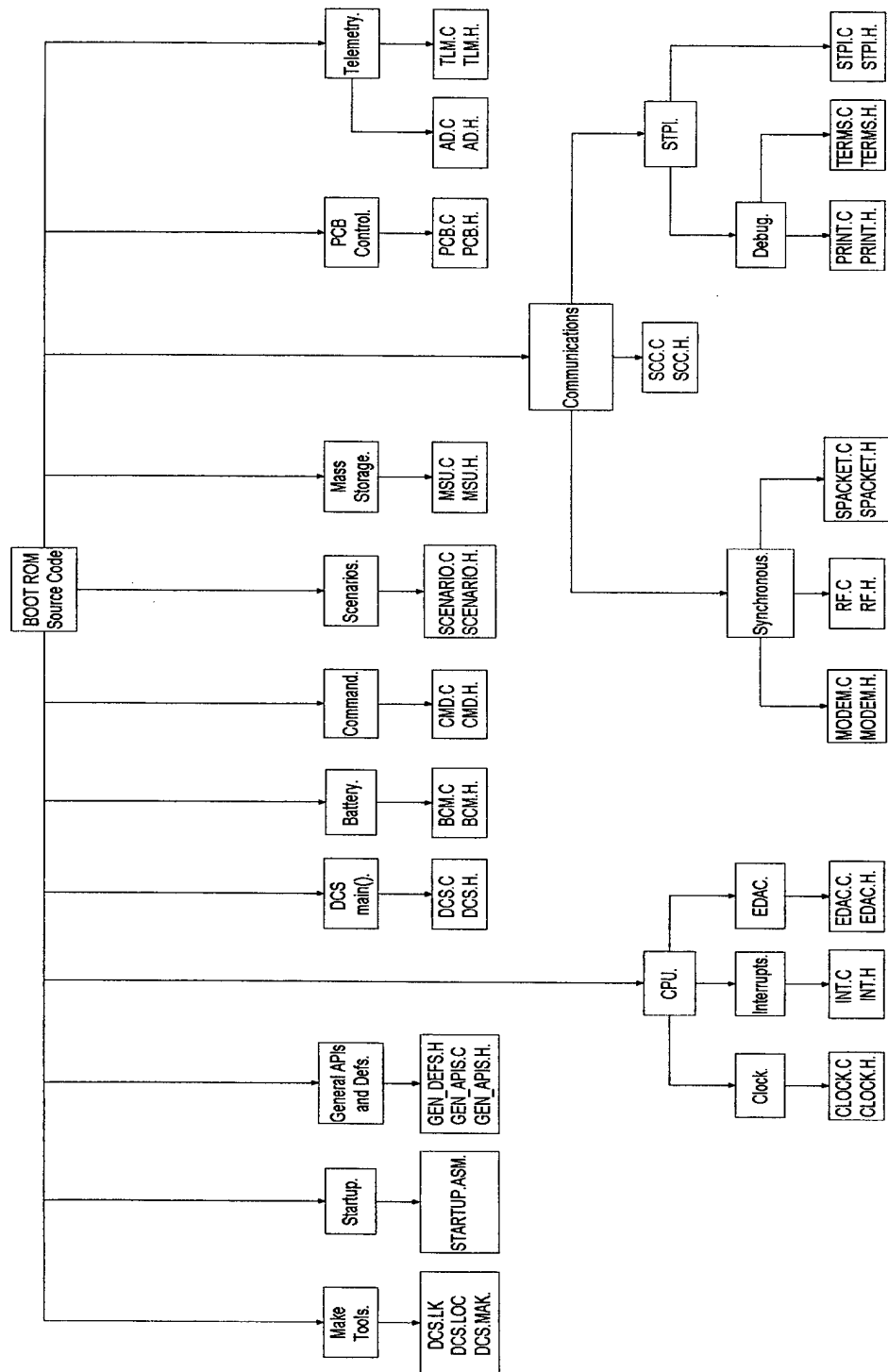
Figure 6. ROM Boot Software File Hierarchy.

IV. SYSTEM CONTROLLER HARDWARE

A.   MICROPROCESSOR

The M80C186XL-10 microprocessor from Intel [Ref. 11, Ref. 12, Ref. 13] provides CPU functionality for the System Controller. This 10 MHz version provides well tested and high-level integration that is very suitable for PANSAT. Within the M80C186 package are the following integrated features:

- CPU core with a CISC instruction set, compatible with 8086/8088 instruction sets.
- Programmable memory (5) and peripheral chip (6) selects, with wait state generators.
- A clock generator, providing three timers.
- Programmable interrupt controller.
- Two channel Direct Memory Access (DMA) controller.

The M80C186XL-10 was chosen for PANSAT for many other practical reasons, such as:

- Military version (providing high reliability at a low cost) of the popular 80186.
- The design team has extensive experience designing Intel-based embedded systems.
- Software development tools were already available for this CPU architecture.

1.   Reset Circuitry and Timing

A simple RC circuit forces the $\overline{RES}$ input of the M80C186XL low for a sufficient time so that the CPU assumes a Reset state. Values of R = 10 kΩ and C = 1 μF create a time constant of 10 msec, assuring the System Controller board of stable power by the time the CPU Reset state is assumed.

2.   Input Clock

A fixed frequency source of 14.7456 MHz feeds the X1 input of the M80C186XL, which in turn divides that signal by two to form a 7.3728 MHz processor and board level system clock for the peripheral devices which use clocks (serial communications controller, memory, and the A/D converter). This is well within the timing limits of the peripherals. Of particular concern are two peripherals. First, the 82C55A will operate up to 8 MHz (it does not use a clock, but rather will read and write up to a speed of 8 MHz). Second, the EDAC unit is designed to work with this timing, and would only allow approximately a 15% faster clock. Faster components would have to be incorporated in the design if the clock rate were to increase (this increase in speed would also cause an increase in power consumption).

3. Interrupts and Direct Memory Access

The M80C186XL has an Interrupt Control Unit, shown in Figure 7, which synchronizes and prioritizes interrupt sources and provides the interrupt vector to the CPU. Hardware interrupts can originate from on-chip peripherals (such as the timers) and from four external interrupt pins.
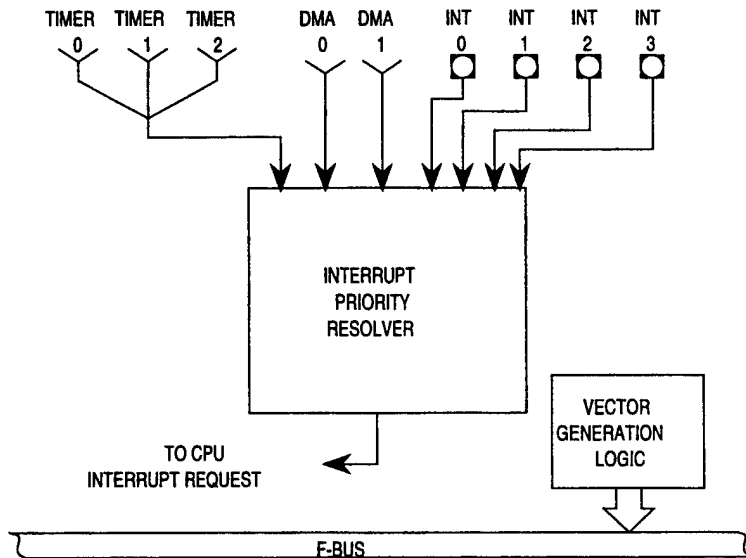
Figure 7. Interrupt Control Unit Block Diagram [from Ref. 12]

External hardware interrupts supported with the design of the System Controller are Interrupt Requests (IRQ) 0 - 3, which are labeled INT0 - INT3, and the DMA requests, which are labeled DRQ0 and DRQ1. IRQ0 through IRQ3 map to absolute interrupt numbers 12 through 15. DRQ0 and DRQ1 map to absolute interrupt numbers 10 and 11. The IRQs are designated to the peripherals of the System Controller that require interrupt support, as shown in Table 1. The DMA interrupts are used with data transfers with the SCC.

Interrupt/DMA Request	Absolute Interrupt	Purpose
INT0	12	SCC (85C30)
INT1	13	A/D Converter (LM12H458)
INT2	14	EDAC Hard Error
INT3	15	EDAC Soft Error
DMA0	10	SCC $\overline{W}$ / REQA (Receive request)
DMA1	11	SCC $\overline{DTR}$ / REQA (Transmit request)

Table 1. External Interrupt Requests.

18

4. Memory and Chip Selects

The M80C186XL supports many programmable memory and chip select signals which reduces the number of external components needed to perform the logic of address decoding to memory and devices. For memory chip selects, there are six select outputs for three memory address areas: upper, midrange, and lower memory. One signal is provided for upper, another for lower, and four for the midrange memory. The range and starting addresses are user-programmable. A unique configuration was chosen for PANSAT. The ROM select is generated using the upper memory chip select. Traditionally, RAM is selected using a combination of the lower chip select and midrange chip selects, depending on the amount of RAM. Since only 512 kbytes of RAM exist, it is possible for the midrange chip select logic to generate all the select signals to the RAM. Since all of the RAM is EDAC controlled, this reduces RAM decoding since only the midrange chip selects need examining, the lower memory chip select can be ignored.

The M80C186XL supports up to seven external peripheral chip selects. However, the System Controller only uses five chip selects ($\overline{PCS4}$ – $\overline{PCS0}$), leaving the remaining chip selects as buffered addresses (A1 and A0) of the CPU. These chip selects access a contiguous block of I/O address space. Each chip select goes active for 128 bytes. The base address can begin on any 1 kbyte boundary, and is programmed to begin with address 0 for PANSAT. Figure 8 shows the chip-select block diagram of the M80C186XL. Table 2 shows chip-select allocation for PANSAT.
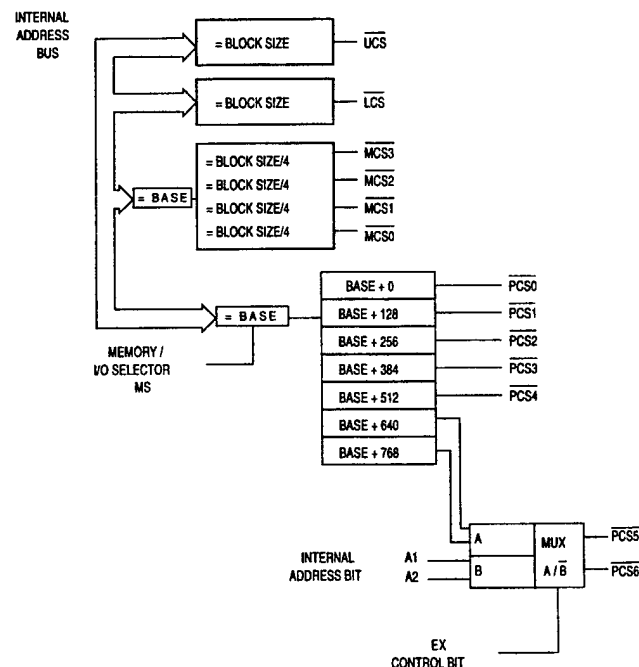
Figure 8.   Chip Select Block Diagram [from Ref. 12].

19

Chip Select	Function	Address Range
UCS	ROM chip select	0xF0000 - 0xFFFFF
MCS3 – MCS0	RAM chip selects	0x00000 - 0x7FFFF
LCS	*Unused.*	*Inactive.*
PCS0	SCC (85C30)	I/O: 0x00 - 0x7F
PCS1	A/D (LM12H458)	I/O: 0x80 - 0xFF
PCS2	PPI (82C55A)	I/O: 0x100 - 0x17F
PCS3	Modem Latch	I/O: 0x180 - 0x1FF
PCS4	PA-100	I/O: 0x200 - 0x27F

Table 2. Chip Selects.

5.   Timers

The M80C186XL contains a timer control unit which has three timers, of which two have external inputs and outputs. The internal timer, Timer 2, is used as a system clock tick generator. The two timers with external outputs, also have external inputs, allowing the triggering of the timers from external sources, independent of the CPU clock. Except for a system clock, the only other need for a timer for the System Controller is a hardware-timed power on switch for the RF output. Since the RF output can be on for up to ten seconds, the two timers are connected in cascade. These timers are used with an external gate to force the RF output off when the timer output expires.

B.   POWER SENSING AND REGULATION

Power sensing and regulation are an important part of the System Controller circuitry. The circuits provide signal isolation between the System Controller and the Peripheral Control Bus when the SC is powered off, monitor switched power from the Electrical Power Subsystem (EPS), and provide regulated +5 V to the System Controller when active.

1.   Power On Detection

The MAX8212 [Ref. 14] is a programmable voltage detector used to sense the powering on of a System Controller by the EPS. Since only the circuitry which isolates a System Controller from the Peripheral Control Bus (PCB) is active at all times, this voltage detector has the responsibility of quickly sensing the power on, removing the signal isolation between the System Controller and the PCB and activating the DC-DC converter, the MAX744A. The output (OUT) of the MAX8212 controls the enabling of the pass gates, 54HC125s, used to isolate signals between the SC and the PCB when the SC is inactive. When power on is not detected, OUT is high, disabling these gates. This circuitry is shown in Figure 9.

20
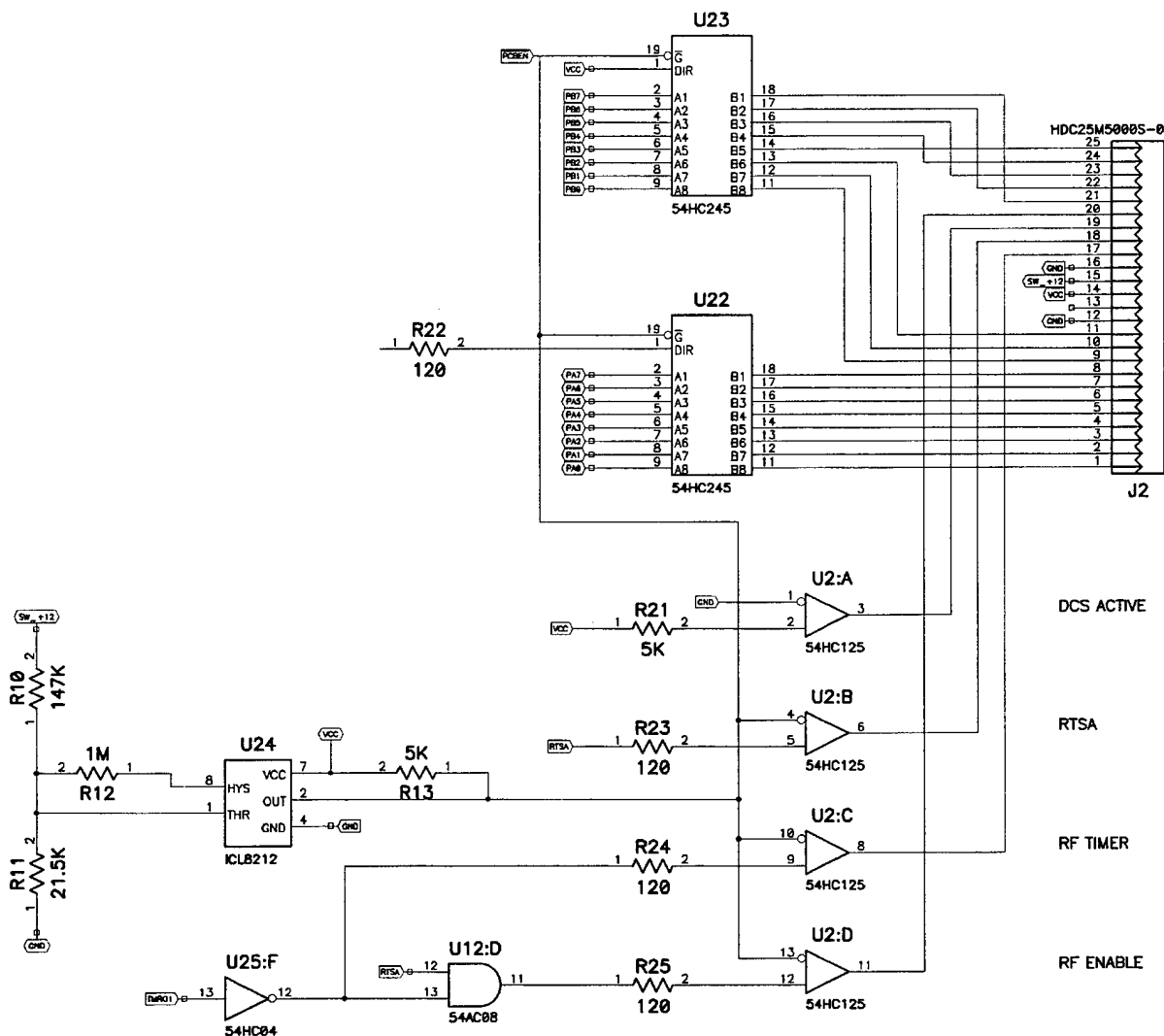
Figure 9. Power On Detect and Signal Isolation.

The resistors labeled 120 are 120 Ω resistors used to reduce current leakage into non-powered devices when the SC is not active.

The MAX8212 uses input hysteresis to set the detect on and detect off voltage levels. These levels are programmed with external resistors. The selection of the resistors was determined by solving for the equations given in the databook (Equation 1) in conjunction with selecting resistors that are available as 1% precision and highly reliable and reducing the source current used by the device. Three resistors are used, R_P, R_Q, and R_S. The detect voltages are V_h, for the high voltage in which the power on detect occurs, and V_l, for the low voltage in which the power off detect occurs.

$$V_l = \left(\frac{R_Q R_S}{R_Q + R_S} + R_P \right)\left(\frac{1}{R_P} \right)(1.15) \quad \text{and} \quad V_h = \left(\frac{R_P + R_Q}{R_P} \right)(1.15) \tag{1}$$

Power from the EPS to a System Controller will be between approximately 10 V to 15 V, depending on whether the solar panels are supplying power or the condition of the batteries, if stored energy is supplying power. A detect on, Vh, voltage of 9.0 V, and a detect off, Vl, voltage of 8.0 V were selected. The resulting resistor values were determined: $R_P = 21.5$ kΩ, $R_Q = 147$ kΩ, and $R_S = 1.0$ MΩ (all 1%). The resulting source current is 53 μA which is within the low operating range of the device.

2. DC-DC Conversion

The MAX744A [Ref. 15] is a current-mode pulse-width modulation DC-DC converter. This device awaits the EPS to provide the System Controller with switched power. The MAX744A takes an input voltage between 6 V and 16 V, an ideal range for the EPS, and converts it to a 5.0 V ±5% output. The device can support a load current of up to 750 mA.

Operating at an input voltage of 12 V, the MAX744A is most efficient (90%) when supplying about 400 mA of current. The System Controller and the Modem unit powered together require about 350 mA (87% efficient). When operating with the Modem unit powered off, a System Controller requires about 70 mA of current (83% efficient). In normal operations, both the SC and the Modem will be powered on continuously.

The MAX744A requires some support circuitry (resistors, capacitors, inductors, and a diode). The resistors and capacitors mostly control the programmable soft-start to ensure an orderly power-up by limiting surge currents. The recommended values for the components were used with the exception of the capacitor, C1, between the Soft-Start input and ground, and resistor, R1, between the Soft-Start input and Shutdown output. Selecting C1 = 0.1 μF and R1 = 511 kΩ while expecting a typical input voltage of 12 V and an output current of 350 mA, a Soft-Start time of 5 msec is expected. Thus, the 10 msec Reset of the M80C186XL-10 is generous. On the output side, a low equivalent series resistance (ESR) capacitor was used to keep the output ripple less than 50 mV peak-to-peak. Additionally, an optional low-pass filter using an inductor and capacitor was added to further reduce the output ripple to about 5mV peak-to-peak. The resulting circuitry is shown in Figure 10.
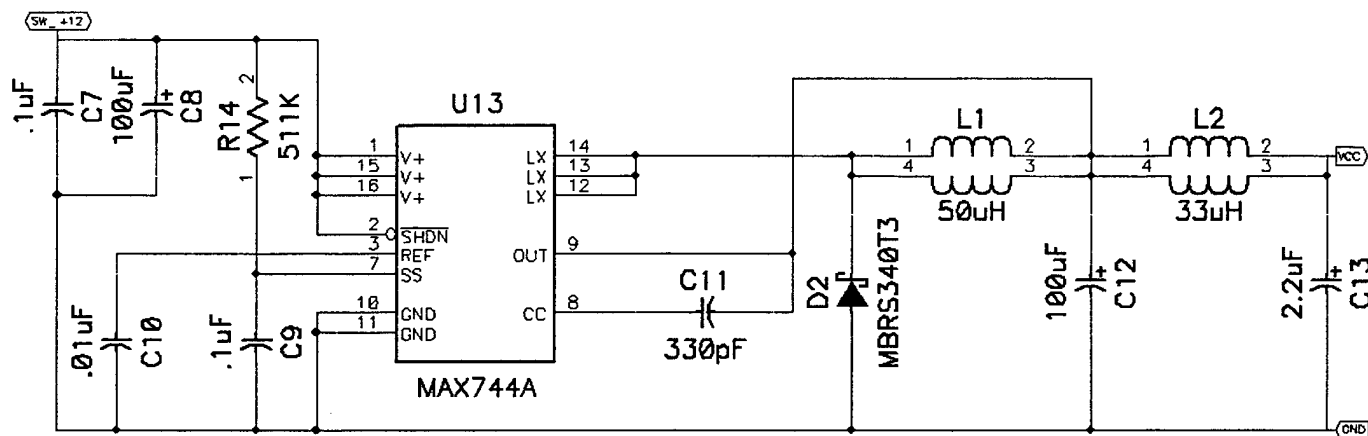
Figure 10. DC-DC Conversion.

C. PERIPHERAL DATA BUS

This section applies to the design of the Peripheral Data Bus which excludes the serial communications control (via the SCC) and the analog to digital converter. The Peripheral Data Bus consists of the circuitry required to implement the Peripheral Control Bus, circuitry to buffer signals that are passed on to peripheral devices (i.e. the 82C55 and the Modem board, and control signals for the EDAC and Modem board power).

The System Controller contains circuitry to provide the driving capability of the Peripheral Control Bus (PCB) which links all of the subsystems of PANSAT with a common control and parallel data bus. This is accomplished using a programmable peripheral interface (PPI, or 82C55A) integrated circuit. In addition to manipulating the PCB, the PPI also controls the signaling to the memory system (the error acknowledge in the EDAC) and the power switch control to the Modem unit.

1. Programmable Peripheral Interface

The 82C55A is a CMOS version of the standard 8255A which provides a general purpose programmable peripheral interface (PPI) [Ref. 16]. There are 24 I/O pins which may be individually programmed in certain logical groups and operational modes.

2. Peripheral Control Bus (PCB)

During satellite startup, the Electric Power Subsystem (EPS) provides +5 V to the PCB. The EPS also cycles all power switches to the peripheral modules so that all are powered off except the +5 V power to the PCB. The PCB must be powered up prior to powering on a System Controller module.

Using the 82C55A in a strobed bi-directional parallel data transfer configuration with hardware handshaking, a digital bus was created with 8 data lines, 6 addressing lines, and two control lines consisting of read and write. In this mode, Port A of the 82C55A is the bi-directional data buffer. And Port B provides an uni-directional address and control buffer. Port C provides the handshaking and three general purpose control lines which are used for controls not on the PCB, but within the System Controller, specifically the Modem Power control and the EDAC Error Acknowledge controls.

Two data transceivers, 54HC245, are used to isolate the 82C55A from the PCB connector. Other signals potentially coming off the SC board onto the PCB are the System Controller Active (SC_A) and the RF Enable (RF_EN) signals. These signals are isolated from the PCB using a quad digital switch, a 54HC125. The voltage threshold detector mentioned earlier assists in the implementation of the PCB isolation circuitry. This circuit disables the two data transceivers and the one digital switch from the PCB in the event that the SC board is powered down.

A 25-pin male D connector is used for connection to the PCB. It contains the eight bits of data, six bits of address, 2 bits of control, +5 V, switched +12 V, grounds, the System Controller Active (SCA) signal, and the RF Enable (RF_EN) signals. Appendix B shows the pin assignments.

3.   Modem Control Interface

The Modem is tightly coupled to the System Controller microprocessor, similar to other on-board peripherals. However the Modem unit is physically contained on another board connected to the System Controller via a 37-pin female D connector. Using data transceivers on each board next to the connectors, buffered address, data, and control signals from the M80C186XL are passed to the Modem. Besides the microprocessor interface, the Modem also shares signals with a Serial Communications Controller (SCC, 85C30) which is described later in this chapter. The signals shared between the Modem and the SCC are synchronous transmit and receive data, and their corresponding clocks. Also included are a signal to deliver the temperature of the Modem board to the System Controller and power and ground. The pin assignments are given in detail in Appendix B.

The Modem operates independently of the microprocessor. However, the heart of the Modem board is the PA-100 demodulator [Ref. 17] which requires configuration by downloading (from the CPU of the SC) parameters as register sets. Occasionally, this processor requires its status to be read by the microprocessor, to determine if a different configuration is needed. Otherwise, the Modem sends and receives synchronous digital data between itself and the SCC on the System Controller without constant supervision by the microprocessor.

Power is distributed to the Modem board in two ways. First, because the Modem board has isolation buffers which disable signals when the board is supposed to be powered off (similar to the PCB

isolation circuitry), there is +5 V delivered to the board at all times via the V_{CC} pins. Furthermore, when the Modem board is switched on by the System Controller, +5 V is delivered to the board via the *+5 V* pins. The switched power is handled by a Power Distribution Switch, the TPS2013 [Ref. 18]. The TPS2013 is a logic controlled heavy capacitive load power distribution switch in the form of an IC. It handles a maximum continuous current load of 1.5 A (more than sufficient for the 250 mA load of the Modem when powered on). This switch is controlled by the Modem Power (MODEM_PWR) signal which originates from Bit 0 of Port C of the 82C55A, and is active low.

4. CPU Signal Isolation and Latching

In order to reduce the signal loading on the CPU and to ease the timing requirements on the I/O digital circuitry, an address latch, a data transceiver, and control signal buffers are used as shown in Figure 11. Since the data signals are bi-directional between the CPU and the I/O peripherals, an appropriate bi-directional data transceiver is used, the 54HC245. This device is enabled using Peripheral Chip Selects of the CPU, $\overline{PCS4} - \overline{PCS2}$, when the Data Enable ($\overline{DEN}$) of the CPU is active. The direction of the transfer is controlled by the Data Transmit/Receive ($\overline{DT/R}$) signal of the CPU. Address signals travel one direction only, from the CPU to the peripherals. Furthermore, since the peripherals themselves decode whether or not they are addressed using the Peripheral Chip Selects, addresses generated by the CPU are latched every time they are generated. The Address Latch Enable (ALE) signal qualifies the addresses. A simple latch, the 54HC573, is used to capture the addresses. Note, since Address 0 (A0) of the CPU is not used for I/O addressing this signal is not passed through the address latch. For peripherals, A1 is the least
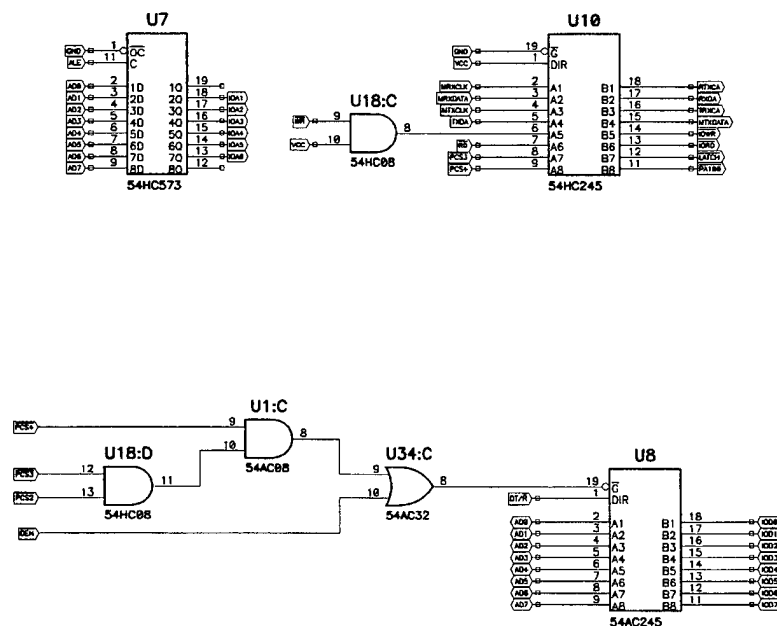
Figure 11. CPU Signal Isolation and Latching.

25

significant address bit. From the CPU point of view, all peripherals are even addressed. Furthermore, the Read ($\overline{\text{RD}}$) and Write ($\overline{\text{WR}}$) signals from the CPU are buffered. Normally, these signals are not buffered since they have adequate driver power. However, these signals are delivered off board to the Modem.

5. Signal Timing to Modem Board

All of the peripheral ICs that integrate with the M80C186XL use an active low write signal, $\overline{\text{WR}}$. The write action occurs on the rising edge of the write signal. This signal is sent through the Modem connector to supply the signal to the Modem board latch as well as the PA-100. The PA-100 however uses the falling edge of the write signal to perform a write action. Unfortunately, data from the CPU only becomes stable at this exact same time. This problem is worsened because the data enabling logic on the System Controller, which controls the enabling and direction of the bi-direction signal driver, delays the delivery of the data signals to the Modem board. As a solution to this problem, a higher speed device is used for the signal driver, the 54AC245 versus the 54HC245. In addition, the decoding logic of the peripheral chip selects, $\overline{\text{PCS4}} - \overline{\text{PCS2}}$, uses a high speed device for the OR gate, the 54AC32, to more rapidly enable the signal driver. The slower speed gate which ANDs $\overline{\text{PCS3}}$ with $\overline{\text{PCS2}}$ poses no problem since during a write to the PA-100 these signals are high and do not change. Finally, $\overline{\text{WR}}$ from the microprocessor is sent through an unused AND gate with the other input held high. This provides a further delay of the write signal to the PA-100.

D. MEMORY

The M80C186XL has a 16-bit wide external data bus. The memory address space is 20-bit, providing up to 1 Mbyte of byte addressable memory locations in the range from 0x0 through 0xFFFFF. The memory address space on the 16-bit data bus is physically implemented by dividing the address space into two banks of up to 512 kbytes, as shown in Figure 12. One bank connects to the lower half of the data bus and contains even addressed bytes, where A0 = 0. The other bank connects to the upper half of the data bus and contains odd addressed bytes, where A0 = 1. Address lines A19 - A1 select a specific byte within each bank. A0 and Bus High Enable ($\overline{\text{BHE}}$) determine whether one bank or both banks are used in the data transfers.

PHYSICAL IMPLEMENTATION OF THE
ADDRESS SPACE FOR 16-BIT SYSTEMS

512 KBYTES

FFFFF
FFFFD
5
3
1

512 KBYTES

FFFFE
FFFFC
4
2
0

A19:1   D15:8   BHE   D7:0   A0
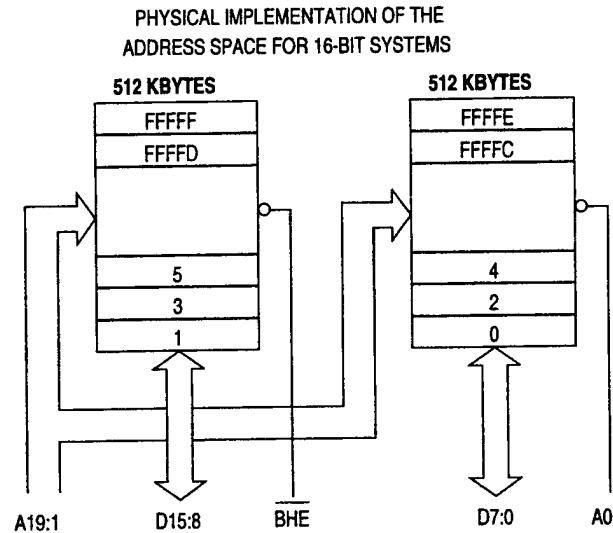
Figure 12. 80186 Memory Addressing [from Ref. 12].

The System Controller Memory includes 64 kbytes of ROM and 512 kbytes of SRAM. Both types of data are addressed from the 16-bit wide external data bus explained in Figure 12. The memory system requires special buffers to temporarily latch both addresses and data during the memory access read and write cycles. Data transceivers reside between the memory circuitry and the microprocessor to reduce the signal loading on the external CPU address and data bus. The block diagram of Figure 13 shows a logical structure of this memory system. On the left are signals that pass between the CPU and the memory system. The AND gates combine the four memory chip selects from the CPU into one signal for the entire 512 kbytes of RAM access. The block labeled ADDR_LATCHES latches a memory address during the beginning of a memory read or write cycle. The blocks in the middle labeled EDAC_MASTER_SM and EDAC_SLAVE_SM contain circuitry that implements the state machines to control the EDAC controller (the ACS630MS). The DATA_XCVRS block isolates the microprocessor data bus from the memory devices, reducing the signal loading of the CPU external address and data bus. The block named RAM contains the SRAM devices as well as the EDAC controller. At the bottom right is the ROM block.
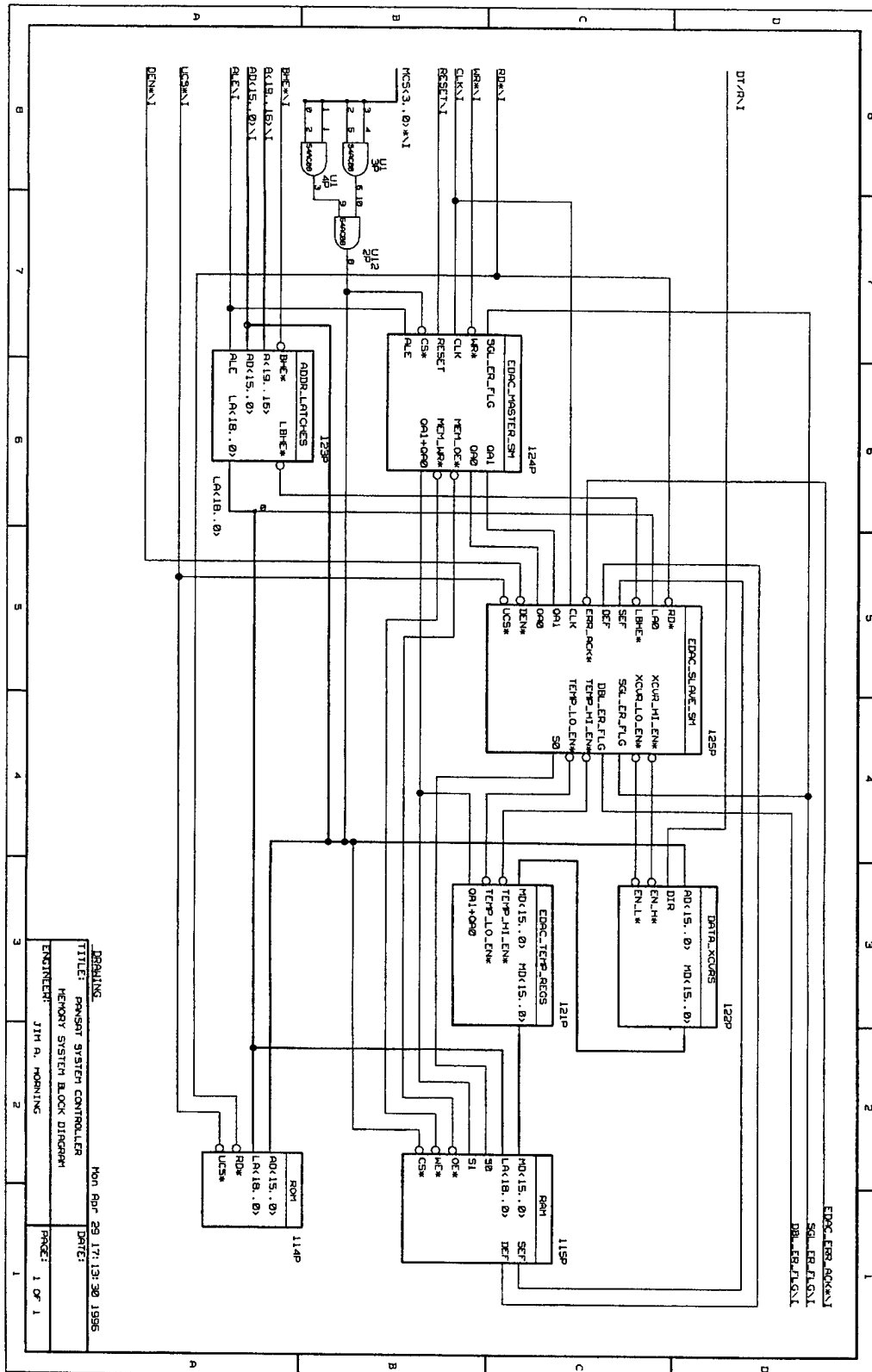
Figure 13. Memory Block Diagram.

28

1.   ROM

PANSAT has 64 kbytes of ROM, composed of two devices which implement the even and odd banks. The M27C256 is a 256 kbit (32 kbyte x 8-bit) CHMOS UV Erasable PROM. This is a 5 V only EPROM requiring low power (200 μA maximum standby, and 30 mA maximum active). This is a military temperature range device with a high degree of protection against latch-up [Ref. 19]. This device provides easy interfacing with the M80C186XL. For development purposes, a low-cost standard version of this device exists. All devices can be programmed easily using the Dataman S4 [Ref. 20].

2.   Error Detection and Correction

Error Detection And Correction (EDAC) circuitry provides single-bit error correction with dual-bit error detection for all of PANSAT's CPU-addressable Static Random Access Memory (SRAM). The error detection and correction is accomplished using a Hamming code to generate a check word for each data word stored in memory. The level of EDAC protection needed determines the number of check word bits per data word. Providing single-bit error correction with dual-bit error detection to an eight data bit word requires five check bits. To provide the same level of detection to a sixteen data bit word requires six check bits.

When a data word is stored in memory, the associated check word is also stored. During a memory read operation the data word and the corresponding check word are retrieved from memory. A new check word based on the data word from memory is generated and compared with the stored check word. If the two check words are identical, the data word is assumed correct; however, three or more bit errors may not be detected. Correctable errors are identified and corrected. Words that are not correctable, but detected as incorrect, cause the error to be flagged.

a. Existing Design

The original Error Detection And Correction (EDAC) circuitry was designed by Oechsel [Ref. 21]. This design is a memory bus controller using a commercially available EDAC IC, the ACS630MS [Ref. 22]. The controller implements a sequential state machine to generate the required control signals for the SRAM (Mosaic's MSM-8256 [Ref. 23]), provides transceivers and latches to isolate the SRAM data bus from the microprocessor local bus, and coordinates the operation of the EDAC IC.

b. Modifications of the Write Back Control

The original design requires that every word accessed from memory be written back to memory, regardless if there is an error or not. This is a good design in that the memory bus controller automatically corrects the error in the data word that is sent to the CPU, as well as the original data word in the SRAM. Although this does not affect the speed performance of the System Controller, since the EDAC

write-back occurs within the memory access cycle of the CPU, it does affect the power used. Writing back to every accessed memory location requires substantial extra memory accesses (i.e. using extra power) as compared to only writing back when memory is written to, or when memory is read and an error is detected and corrected. As a result, the write back control logic was modified to only write to memory when memory is being written to by the CPU, or when an error is detected and corrected following a memory read access.

The control signal that causes SRAM to write a data word is $\overline{MEM\_WR}$. Figure 14 shows a Karnaugh Map and the resulting equation that eliminates the write back of data during read cycles when the single error latch is not set.

$$\overline{MEM\_WR} = \overline{QA1} + QA0 + \overline{SGL} \cdot \overline{WR}$$
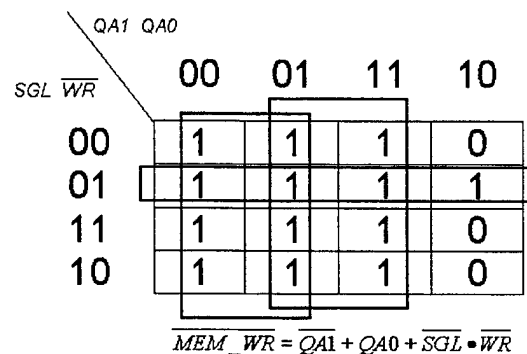
Figure 14. Memory Write Back Logic.

Using unused gates from the remaining glue logic needed to implement the memory bus controller, Figure 15 shows the circuitry which performs this write back function.
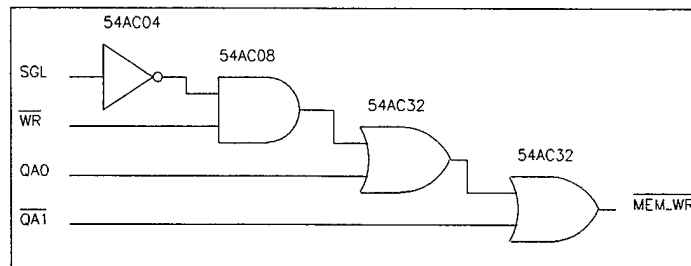
Figure 15. Memory Write Back Circuit.

c. Modification of the Reset Circuitry

The original design used an inverter to reset the master state machine. As an alternative, the flip-flop's reset input is directly connected to V_{cc}. This is possible because by the time the M80C186XL has completed a reset during power up, it has asserted the reset signal during the power up, and the flip-flop has already reached the initial standby state (00).

d. Modification of the EDAC Error Acknowledge

During the development of the original EDAC circuitry, for simplicity one of the peripheral chip selects of the M80C186XL was used to acknowledge a hard or soft error. Since all the peripheral chip selects have I/O addressing responsibilities for the System Controller, an alternative was needed. The 82C55A described above has control bits available on its Port C. Bit 2 of this port is used as the $\overline{ERR\_ACK}$ signal to the EDAC.

e. Modification of the Transceiver Enables

The original design of the EDAC circuitry connected the ROM directly to the microprocessor data bus. However, it is desirable to isolate the ROM from this data bus. To use the same transceivers that are used for the SRAM, additional logic was needed at the state machine outputs which control the enabling of these transceivers. Allowing a valid combination of Data Enable ($\overline{DEN}$) and the Upper Memory Chip Select ($\overline{UCS}$) with the state machine's output (which is valid for a SRAM access), the following circuitry in Figure 16 shows the modification.
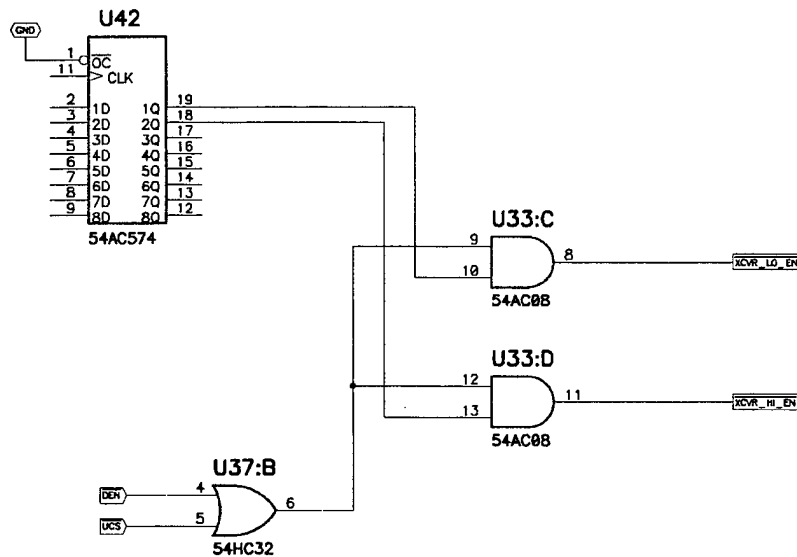
Figure 16. Memory Transceiver Enable Logic Circuit.

E. ANALOG-TO-DIGITAL CONVERSION

The System Controller is responsible for converting all analog signals pertaining to satellite telemetry (voltages, currents, and temperatures) to a digital counterpart. The SC accomplishes this using a dedicated analog-to-digital converter which is tightly coupled to the microprocessor bus. In addition, since incoming analog signals arrive whether or not the SC is active, an analog switch is used in conjunction

with the PCB isolation circuitry to automatically isolate these signals from a non-active SC. A low-pass filter removes the high frequency components associated with the Electrical Power Subsystem DC-DC conversion (and other high frequency noise generated within the satellite).

1.   A/D Converter

National Instrument's LM12H458 [Ref. 24, Ref. 25], a 12-bit data acquisition system, is used to provide analog-to-digital conversion to the System Controller. This device provides single-ended or differential self-calibrating conversion with its own sample-and-hold. Multiple conversions remain in a 16-bit, 32 register FIFO buffer. An internal 8-word RAM stores the conversion sequence for a program that can acquire independently and convert up to eight acquisitions through the eight-input multiplexer. The LM12H458 runs on +5V and an input clock of up to 8 MHz. In standby mode, the device consumes a negligible 40 µA (maximum). Input signals can range from ground to +5.0 V.

For PANSAT, the LM12H458 is configured to interface with the M80C186XL in a 16-bit data mode. The inputs of the LM12H458 can be single-ended or differentially acquired by use of programmable input configurations. In differential mode, the off-board analog signal with its accompanied ground are converted as a positive signal with reference to this accompanied ground signal. Otherwise, the ground of the LM12H458 can be used when converting in the single-ended mode. For the temperature sensor ICs, local to the SC and the Modem board, these signals are delivered single-ended, and thus conversion uses this mode only.

The LM12H458 is capable of interrupting the microprocessor and uses this to signal end-of-calibration and end-of-sequence (acquisition finished) conditions. Although DMA is available, it is not used, sacrificing the M80C186XL's two DMA channels exclusively for the Serial Communications Controller.

2.   Analog Switch

Immediately after entering via the connector on the SC, analog signals pass through an analog switch. The DG411, a monolithic quad SPST CMOS analog switch from Harris [Ref. 26], provides this feature. This device has a low On-Resistance (less than 35 Ω), and is a very low power device consuming approximately 5 µA. As mentioned earlier, incoming analog signals arrive whether or not a System Controller is active. Thus, this analog switch is used to isolate the incoming signals from a non-active SC. Three of the four channels of the DG411 are used to handle the off-board signals from the Electrical Power Subsystem and the two Temperature MUXing Subsystems. When the SC is powered off, this device is powered via the Peripheral Control Bus and the switches are forced open. Upon being powered on, the SC is designed to automatically close these switches, allowing the external analog signals to enter.

3. Voltage Clamping and Low-pass Filter

Studies performed on a prototype of the Electrical Power Subsystem indicated high frequency components in incoming signals at frequencies around 4.5 kHz, attributed to the switching power supply's DC-DC converter. Furthermore, unaddressed multiplexers from the Temperature MUX subsystems and the EPS create signals of about 9.5 V. After advancing through the analog switch, an analog signal is potentially clamped if over 5.0 Volts using a 5.1 V Zener diode, 1N751, and then enters a passive low-pass filter using an RC circuit where R = 1 kΩ and C = 0.1μF. The filter is shown in Figure 17. The filter provides a -3dB break frequency at 1.5 kHz.
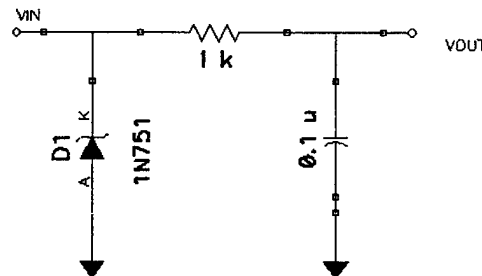
Figure 17. Voltage Clamping and Low-pass Filter

4. Wiring

As mentioned earlier, the acquisition of off-board analog signals can be performed differentially or single-endedly. Undesirable potential ground-loops may exist within the satellite electrical systems, causing differing ground reference points. In order to be able to compensate for this potential problem, the analog input system was designed to allow the following configurations. Off-board analog signals coming into a System Controller are sent using a twisted-pair wire set. One signal in the twisted pair is the so-called positive signal which carries the signal of interest for conversion. The other wire in the pair is the ground from the electronic circuits that are responsible for generating the signal for conversion.

For the analog signals entering the System Controller, the board was designed to allow one of the following configurations. The ground signal from a twisted-pair can be ignored by using the single-ended mode of the LM12H458; this feature is software configurable. Additionally, a twisted pair ground signal can be isolated from the System Controller circuitry by not allowing the signal to pass beyond the connector; this is accomplished using a jumper.

5. Connector

A nine-pin male D connector passes incoming analog signals to the System Controller board. The pin-out is given in Appendix B.

33

6. Temperature Sensing IC

Although not part of the A/D conversion circuitry, a temperature sensing IC is used to provide a temperature sensor for the System Controller. A single-supply centigrade temperature sensor, the LM50C [Ref. 27], is a precision IC temperature sensor that can sense a range of -40° C to $+125^\circ$ C. The output voltage is linearly proportional to the temperature. It consumes very low power with a quiescent current of less than 130 μA while operating at $+5$ V. This sensor delivers its signal into the Analog-to-Digital converter.

F. SERIAL COMMUNICATIONS

The System Controller contains circuitry that allows both asynchronous and synchronous serial communications. The asynchronous mode provides an RS-232 compatible interface suitable for connecting to a standard computer serial interface, such as the COM port on a personal computer. The asynchronous RS-232 port is used for this purpose. In the synchronous mode, the System Controller is able to communicate with PANSAT's Modem. The synchronous digital data passes through the Modem as the interface to the RF unit to provide the BPSK and spread-spectrum communication modes (either at 9846 bits per sec for spread-spectrum mode or at 78.125 kbits per sec for the narrow band mode).

PANSAT will transmit and receive at a center frequency of 436.5 MHz using direct sequence spread spectrum (DSSS), differentially encoded phase shift keying (DPSK) modulation. The communications data rate will be 9842 bits per second with a chipping rate of 1.25 mega chips per second. The chipping sequence is a pseudo random noise sequence produced from a 7 stage linear feedback shift register. In addition, the satellite will have a narrow band high data rate channel with a data rate of 78.125 kbits per second.

1. Serial Communications Controller

The heart of the serial communications circuitry is the AM85C30 [Ref. 28], a serial communications controller (SCC) which provides two serial channels that are independently configurable for asynchronous or synchronous transmission modes with separate transmit and receive clocks. The SCC functions as a serial-to-parallel and parallel-to-serial converter with the CPU. It is software configurable by the M80C186. The SCC interrupts the microprocessor using the M80C186's Interrupt Request 0, INT0. The signal, $\overline{\text{INT}}$, is active low on the SCC and is thus inverted to be compatible with the active high INT0. A pull-up resistor keeps the output of the inverter low during the SCC initialization, thus producing no interrupt to the CPU. This circuitry is shown in Figure 18.
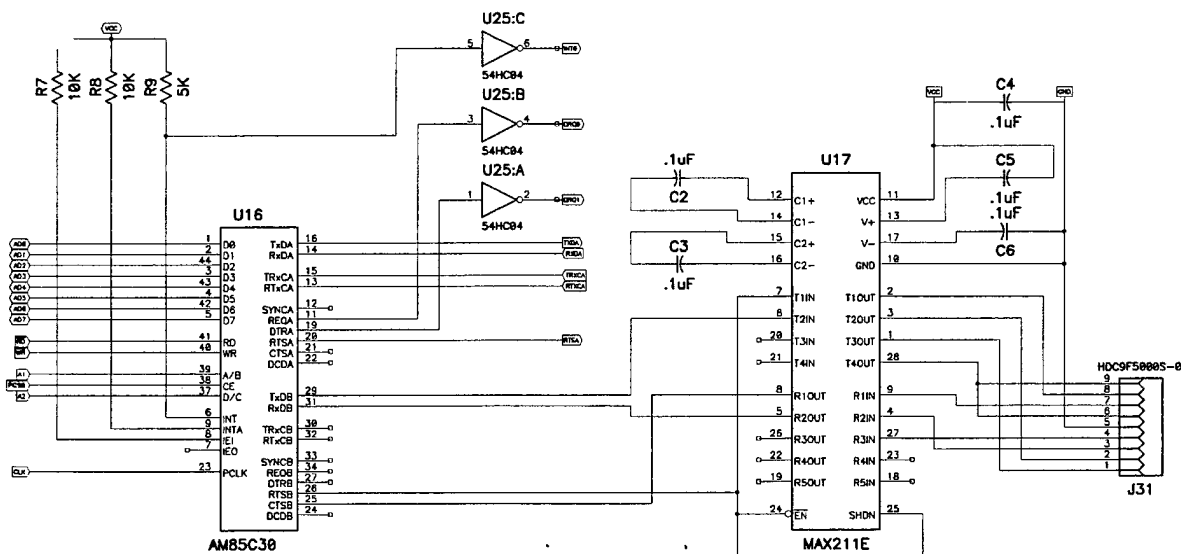
Figure 18. Serial Communications Circuitry.

Channel A of the SCC is designated the synchronous channel for serial communications between the CPU of the System Controller and the Modem board. This channel is configured to send and receive data using external clocks (TRxCA and RTxCA) which are generated by the Modem board. While operating in the narrow-band mode, the modem supplies transmit and receive clocks of 78.125 kHz (1 byte every 102 μsec). In the spread-spectrum mode, the modem supplies transmit and receive clocks of 9.842 kHz (1 byte every 813 μsec). Furthermore, the transmit and receive data signals (TxDA and RxDA) pass through to the Modem board. Channel A is also configured to assert the Request To Send (RTSA) signal which acts like a Push To Talk function found on radio transmitter equipment. This signal will be used to drive the RF unit when transmitting. Channel A has some additional signals used specifically with the CPU for DMA. The signals for a Receive Request ($\overline{W / REQA}$) and for a Transmit Request ($\overline{DTR / REQA}$) are tied directly to the CPU's DMA interrupt request inputs. The SCC uses active high assertion levels for its DMA requests: $\overline{W / REQA}$ for DMA0 and $\overline{DTR / REQA}$ for DMA1. Inverters are used as in the case of the interrupt request, shown in Figure 18. This hardware configuration is capable of full-duplex communications. However, the hardware will only operate in a half-duplex due to the allocated frequency bandwidth in which PANSAT will operate. This synchronous channel is programmed to operate in the Synchronous Data Link Control (SDLC) mode which applies a cyclic redundancy check (CRC) using the CCITT CRC-16 algorithm with a CRC seed of 0xFFFF. Thus, all synchronous data frames which fail the CRC are flagged as invalid and are ignored by software.

Channel B of the SCC controls the asynchronous serial communication between the CPU of the System Controller and a separate Computer. This channel hosts the Serial Test Port Interface which is used

35

only for ground-based operations before launch. It is configured to send and receive data using an RS-232 driver, describe below. This channel also drives Request To Send (RTS) and Clear To Send (CTS) signals which are used in the RS-232 handshake. A common clock, generated internally by the SCC as derived from the peripheral clock, is used to generate a data rate of 9600 bits per second. $\overline{RTSB}$ is used to drive the enable of the RS-232 driver discussed below.

2. RS-232 Drivers and Receivers

The MAX211E [Ref. 29] has multiple RS-232 line drivers and receivers that are suitable for communications in a harsh environment. Although the asynchronous communication mode will not be used while in flight, this mode will be used extensively during development, integration, and testing. This port is a main source of external physical coupling with other electronics, and a potential source of damage to electronics within the satellite. Thus, the justification for the device which has ±15 kV electrostatic discharge protection. In RS-232 terminology, the System Controller acts like a Data Communications Equipment (DCE) and the separate computer is a Data Terminal Equipment (DTE). The signal assignments, described in Appendix B, follow this convention. The MAX211E is enabled by using the $\overline{RTSB}$ output of the SCC. When disabled (when not used for development purposes), the MAX211E uses a maximum supply current of 50 μA.

3. Connector

A 9-pin female D connector is used to pass the signals from Channel B used for asynchronous communication between the System Controller board and a separate computer. The pin-out configuration was chosen to be compatible with a personal computer COM interface so that a null-modem interface (crossing of send/receive signals) is not necessary. Thus, a straight through cable is used to connect the two systems. The signals, shown in Appendix B, are from the point of view of the System Controller.

This concludes the discussion of the System Controller hardware design. This hardware supports the development and operation of software that control the spacecraft's electronic modules by using the satellite's general purpose computer. The device drivers for the System Controller hardware and the electornic modules of PANSAT are described in detail in the following chapter.

V.   SYSTEM CONTROLLER SOFTWARE DRIVERS

A.   DESCRIPTION

The Spacecraft Software for PANSAT is contained within the System Controller as partially embedded in ROM and additionally uploaded (from Earth) at the time of system bootup and when software updates are desired. Within PANSAT lie two (2) System Controllers, each with identical Spacecraft Software ROM.

Operation of the spacecraft software consists of the completion of the hardware initialization and the creation of a runtime environment suitable for higher-level layers of software. Operation then consists of monitoring and charging the batteries while establishing primitive communication with the PANSAT Ground Station at NPS to begin the upload of the main operating system kernel and the secondary loader. Thereafter, a higher level protocol will then upload BAX (the BekTek AX.25 protocol handler), and another telemetry collector. AX.25 is a data-link layer protocol designed and used by the Amateur Radio community to perform digital packet radio communications [Ref. 30]. Using AX.25, the File System and user services software are uploaded. Finally, the spacecraft is then ready for ggeneral use.

The software is written in C with 80186 assembler for critical code and the initial Bootstrap software. The spacecraft software incorporates the Spacecraft Operating System [Ref. 31] (SCOS), to provide an Application Program Interface (API) to simplify the job of writing multi-tasking applications. The Spacecraft Software also incorporates BAX [Ref. 32], to provide another API to simplify the use of AX.25 for Spacecraft Software tasks.

The spacecraft software involves the porting and integration of SCOS and BAX. It also consists of various device drivers to interface with the hardware systems of the spacecraft, the collection and saving of telemetry, and an interface capability with possible experiment payloads via an RS-232 interface. Furthermore, the spacecraft software contains a store-and-forward mail system which uses the services of the File Transfer Level 0 (FTL0) protocol [Ref. 33]. Administration and system software and parameter update capabilities are also part of the spacecraft software.

The discussion of software development that follows concentrates mainly on the code responsible for the Bootup software which is embedded in ROM; this is the core of PANSAT's device driver software. Therefore, when referencing the generic term *software* within the following text, the reference will be to this Bootup software which contains the core device drivers and boot loader and not SCOS, BAX, nor the high-level user services.

37

B. HIERARCHY AND MODULE RELATIONSHIPS

A flow of software control from Reset to the completion of the Final Loader can be viewed below.

Spacecraft Module	Spacecraft Function			Ground Station Module	Ground Station Operations
STARTUP	• SC hardware initialization and EDAC setup. • Setup C Runtime environment.	R O M		CONTACT	• Send message to PANSAT, granting PANSAT authority to send a status message. • Listen for the spacecraft status message.
BLOADER	• Check batteries and charge. • Listen for NPS Ground Station. • Performs initial upload of software from NPS Ground Station: KERNEL and SLOADER, transfers control to O.S. kernel	R O M		SCLOAD CONTACT	• Begin by sending messages to PANSAT, granting authority to send a status message, and listen for the spacecraft status message. • Use SCLOAD to prepare images of the O.S. kernel (KERNEL) and the secondary loader (SLOADER). • Use CONTACT to send binary images.
SLOADER	• Performs secondary upload of software from NPS Ground Station: BAX, FLOADER, CMDTLM.	R A M		SCLOADH	• Use SCLOADH to prepare (and send) images of AX.25 (BAX), Final Loader (FLOADER), and the Command/Telemetry (CMDTLM) software tasks.
FLOADER	• Performs AX.25-assisted uploads of remaining software from NPS Ground Station: FS, FTL0, BBS, other tasks. • Used for further software uploads.	R A M		SCLOADH	• Use SCLOADH to prepare (and send) images of the File System (FS), File Transfer Level 0 (FTL0), the Bulletin Board Server (BBS), and other tasks.

Table 3. Relationships Between Software Modules.

C. STARTUP

The name Startup refers to a software source code module. This module is software composed in 80186 assembly language and is called *startup.asm*. After a System Controller receives a reset, it is the software contained within this module which is invoked first, containing the bootstrap code at the hardwired absolute memory location of 0xFFFF0 which is contained within the ROM.

Startup has several goals. The first is to initialize any hardware peripheral on the System Controller board itself, so that the peripheral is placed in a known state (although not necessarily operable). This activity is called Hardware Initialization. Furthermore, Startup checks and clears all of the system

RAM which is controlled by the EDAC. Once the system RAM has been checked, data which is used by code composed in C (both embedded within the ROM) is relocated to RAM. This allows read/write access to variables referenced by C code. In order to support high-level floating point software statements, a floating point emulator (software-based) is used and must be set up before use by any higher-level software modules. Finally, a runtime environment becomes operable for the subsequent software modules, all composed in C, to operate normally.

1.   Hardware Initialization

Hardware Initialization begins by programming the M80C186XL's [Ref. 13] internal peripheral interface registers. These registers are grouped into a block with contiguous addresses and are by default referenced via an I/O reference beginning at 0x0FF00. This default reference was kept. The M80C186XL contains multiple Chip Selects to internalize chip select generation for ROM and RAM. After a Reset, the Upper Memory Chip Select (UCS) will correctly select when referencing the top-most 16 bytes of system memory, 0x0FFFF0 through 0x0FFFFF. Therefore, the first initialization performed is to change the range of addresses that the UMCS will generate. In the case of PANSAT, where the ROM is 64 kbytes in size and occupying the top-most 64 kbytes of system memory, UMCS is configured to start selects at 0xF0000 for a block size of 64 kbytes.

```
cli
mov         dx, 0FFA0h       ;upper memory chip select
mov         ax, 0F038h       ;start of EPROM F000:0h, 64K
out         dx, ax
jmp         far ptr START    ;START 0F0000h
```

The bottom-most location of the 64 kbytes of system ROM is named START. At this location begins the remaining startup code. This code continues the programming of the peripheral interface registers. For memory and peripheral chip selects (LCS, MCS, PCS) the table below describes the configuration of these important selects. By not programming the LCS (not performing a read or write operation to the configuration register), this chip select remains inactive.

Register	Register Address	Register Value	Setup Description
PACS	0xFFA4	0x0000	Peripheral Chip Select base: PCS0 = 0x0000, each block is 0x80 in length. Bus Ready must be active to complete bus cycle, no wait states inserted in the bus cycle.
LMCS	0xFFA2	*Not Applicable.*	Lower Memory Chip Select: *Unused.*
MMCS	0xFFA6	0x41F8	Mid Memory Chip Selects: 0:0 to 0x7FFF:F (512K block).
MPCS	0xFFA8	0x2000	Peripheral Chip Select: Starting at 0, PCS5/6 latch A1 & A2, PCSx go active for I/O bus cycles, requires bus ready be active to complete bus cycle (applies to PCS4-6), no wait states inserted for PCS4-6.

Table 4. Memory And Peripheral Chip Selects.

Before hardware peripherals of the system controller are initialized, Startup also configures the M80C186XL Timer0, used as a system clock, and the interrupt controller to temporarily disable all interrupt sources. Startup then configures the hardware peripherals. The PPI configuration ensures the device is setup for bi-directional data exchange with the correct handshaking (discussed in more detail later), and places certain control lines to their correct assertion levels. The Modem Power control bit (active LOW) is turned off (programmed to a 1), the EDAC reset line is set on (programmed to a 1), and the RF enable is turned off (programmed to a 0). After the PPI, SCC configuration follows, placing the device in the correct modes for synchronous communication for Port A, and asynchronous communication for Port B.

2.   Memory Check and Clear

During Startup, before interrupts have been enabled and before a stack has been created and activated, the entire RAM is briefly checked for errors and in the same process the memory cells are cleared, removing the possibility of later RAM accesses causing EDAC soft or hard errors. First a write followed by a read of the data 0x55 and 0xAA is performed in the entire RAM. The purpose of this alternating bit pattern test is to determine if there are data bus problems, bad devices, or chip select failures. However, this test only tests the ability of a device to hold data, but not that the devices are being addressed correctly. Thus, another test is performed which writes unique data to each location and then reads back. The unique data will be a 257-bit repeat test, e.g. cell 0 will get the value 0, cell 1 gets 1, ... Cell 256 gets 256 (modulo 256 = 0), cell 257 gets 257 (modulo 256 = 1). Then the pattern repeats so that cell 258 gets 0, cell 259 gets 1, etc.

In the event that memory cells fail a test, the software system will attempt to map out the bad cells if possible. However, if errors occur in the interrupt vector table, there is no mapping solution. The System Controller will force itself to fail to update the EPS watchdog timer, thus shutting itself down.

3.   Data Relocation

Since all data (constants and variables) of any C software module are embedded within the ROM and may at some later point be relocated to RAM, data relocation is a necessary step for proper configuration of the C runtime environment. True constants (never modified data references) may remain in ROM. However, any variable which is not created on the C runtime stack, must be relocated and possibly initialized in RAM. The data relocation performed for PANSAT is based upon standard techniques [Ref. 34].

As already discussed, the 80C186 hardware architecture uses the concept of segments to organize physical directly-addressable memory. Software also uses the word segment. Segment, in the context of software, can refer to a physical 64 kbytes segment of the 80C186, or also a logical grouping of data or

code. DGROUP is a group of like data segments particular to the Microsoft C Compiler for code generation of a memory reference model called the Small Model. Small Model programs are by definition fixed in data segment size to a physical segment size of 64 kbytes (directly related to the 80C186's architecture) and fixed in code segment size to also 64 kbytes. However, the data and code segments occupy separate physical segments, and thus provide up to 128 kbytes of code and data. As a convenience to the compiler and linker, data groups are created to collect similar types of data references [Ref. 35].

Data relocation begins by clearing any initialized data area in the DGROUP group. Initialized data is any data that is a variable, not created on the runtime stack, and is given a specific value at the time of code composition. Thus, it is necessary that this prior, initialized, value given to this variable must be preserved. As all of these initialized variables are relocated to RAM, their values are preserved.

Data relocation continues with uninitialized data area in the DGROUP group. Uninitialized data is any data that is a variable, not created on the runtime stack, and is given no value at the time of code composition. Thus, it is assumed that any prior value given to this variable is not of importance. As a matter of consistency, all of these uninitialized data references are relocated to RAM and forced to the value of zero.

4. Floating Point Emulation

Although the 80C186 contains hardware and software hooks to allow a companion floating point coprocessor, namely the 80C187, this option was not used for various reasons. Primarily, in the style of attempting to make a lean system controller, floating point arithmetic is considered a luxury and minimizing its use is practical. Also, this coprocessor, like many floating point processors, is a heavy power user consuming about 50% more power than the M80C186XL alone. Furthermore, using software, floating point arithmetic can be emulated. This emulation occurs at a great sacrifice of CPU cycles since some floating point instructions can take hundreds of more CPU time when emulated. However, floating point arithmetic is minimized.

The software tools used for linking and absolute address location provide a ROMable floating point emulation library. Within the Startup are subroutine calls to the floating point library initialization routines. In cooperation, the C compiler when compiling translates all floating point expressions to a series of calls to subroutines.

5. C Runtime

Prior to the transfer of control to the code of the collection of C-composed modules, appropriate hardware interrupts are masked on, e.g. EDAC and Timer2. Furthermore, a runtime stack is created so that an area of RAM is set aside for stack-related operations. The stack for the C runtime environment is 4

kbytes of EDAC-controlled RAM, located just after the interrupt vector table. The stack creation actually occurs just prior to the calling of the floating point emulation routine. Transfer of control is accomplished by calling the main C software module, named *main()*. If ever main() should terminate, the code following this call turns off interrupts (not allowing the clock and Watchdog support software to operate) and finishes with a HALT instruction.

D. CPU SUPPORT

1. Timers and Interrupts

a. Timers

The Timer/Counter Unit, integrated within the M80C186XL [Ref. 12], supports three (3) independent 16-bit counter/timers. These timer/counters are used to generate a system clock implemented by the operating system, and other time-dependent functions. Table 5 shows the use of each timer.

Timer	Function
0	Transmitter time-out (feeds clock to Timer 1)
1	Transmitter time-out (gates RF Enable)
2	System Clock (operating system)

Table 5. Timer Allocation.

Cascaded, Timers 0 and 1 can provide a maximum timer of 38.84 min (each timer is updated every ¼ CPU clock which is 0.542535μsec, $2^{16}*2^{16}*0.542535$μsec). Timer 0 is programmed to run continually using PCLK as a clock and given a maximum count value of 2^{16}, providing a 35.5 msec clock for Timer 1. Timer 1 uses Timer 0 as its input clock. Using the dual count mode as a one-shot timer, Timer 1 is programmed to stay low when enabled for 10 seconds using a Count A value of 1 and a Count B value of 281. The output of Timer 0 is inverted before it is gated and passed off the System Controller board to the RF unit, providing an active high signal. Ten seconds is a suitable length of time to enable the transmitter, and yet provide an automatic turn off mechanism.

Timer 2 is programmed to interrupt the CPU at a frequency of 60 Hz, providing a tick counter to implement a system clock. This timer uses a maximum count of 30720 in a continuous count mode. The interrupt service routine counts the interrupts to maintain a second counter. Two API functions provide the ability for software to read and set the second counter, based on UTC from 1 January 1970. Thus, when PANSAT is reset, it begins with a date of 1 January 1970 until otherwise programmed by the NPS ground station. A 32-bit second counter allows dates until 2106. Within the clock ISR is the chaining to the EDAC RAM Wash subroutine which is explained in more detail later in this chapter.

Function Name	Description
get_time()	Get UTC time in seconds (elapsed time in seconds since 1 Jan 1970).
set_time()	Set UTC time (elapsed time in seconds since 1 Jan 1970).

Table 6. Clock API functions.

b. Interrupt Priority Structure

The Non-Maskable Interrupt (NMI) is the highest priority interrupt and will be disabled via hardware to prevent this event from occurring unless the Error Detection and Correction (EDAC) circuitry is enabled. Maskable interrupts are the most common way to service the external hardware interrupts. Globally, software can enable or disable the maskable interrupts. Maskable interrupts have priorities among themselves which are determined by the programming of the interrupt control unit and the support circuitry. Exceptions occur when an unusual condition prevents further instruction processing until the exception is cleared. Software interrupts are generated by software using the *INT n* instruction. The M80C186XL handles exceptions and software interrupts in the same way as hardware interrupts and are of lower priority than the maskable (hardware) interrupts. All unused (undefined) interrupt vectors are initialized to point to 0x0FFFF0, the Reset vector, so that if an undefined interrupt occurs, the system will perform a reset. Table 7 shows the interrupt vector use.

Interrupt Vector Number	Interrupt Function
0	Divide-by-zero
1	Single Step
2	NMI
3	Breakpoint
4	Overflow
5	*Undefined*
6	Invalid Opcode
7	Escape Opcode
8	Timer 0
9	*Undefined*
10	DMA0
11	DMA1
12	INT0 (SCC - 85C30)
13	INT1 (A/D - LM12H458)
14	INT2 (EDAC Hard Error)
15	INT3 (EDAC Soft Error)
16	*Undefined*
17	*Undefined*
18	Timer 1 (Transmitter time-out)
19	Timer 2 (SCOS clock tick generator)
20 - 255	*Undefined*

Table 7. Interrupt Vector Allocation.

2. EDAC (Setup and RAM Wash)

Error Detection And Correction (EDAC) memory will cover all addressable Spacecraft Software system RAM and will ensure the state of memory which is within the data and code addressing range of the M80C186XL. The memory decoding is designed such that all of the 1/2 Megabyte of direct addressable memory, from 0 to 0x7FFFF, is EDAC RAM for the CPU. With the EDAC circuitry enabled, all memory cycles to RAM, either byte or word size, will be intercepted by the EDAC. Each write generates error correction patterns which are recorded as well as the data. Each read retrieves the data as well as the correction patterns to determine if the data read is correct, is correctable (soft error), or is not correctable (hard error). The EDAC will indicate these two types of error to the M80C186XL via interrupts. The soft error produces a maskable interrupt, INT3. The hard error produces a maskable interrupt, INT2.

There are two errors associated with a RAM memory cycle. The first error is called a soft error and is completely correctable. With a soft error, the EDAC is able to reconstruct the correct bit pattern of the byte or word cell. The EDAC will respond to a soft error by sending the correct byte or word and then asserting an interrupt. The interrupt service routine (ISR) is responsible for remembering at what time the soft error occurred. The second error is a hard error which is not correctable. In the event of a hard error, the EDAC asserts another interrupt. The soft error service routine will first save the necessary registers on

the stack. Then a message indicating that a soft error occurred will be sent to the Event Logger. The soft error service routine will then restore the registers and return from interrupt service. The hard error service routine must assume that any RAM cell is corrupted and cannot necessarily be corrected, because the location cannot be determined. Therefore, this service routine will suspend all interrupts, place all the hardware systems into an idle or off state, and then reboot the M80C186XL by jumping to the boot vector.

a. Initial RAM Clearing

Upon microprocessor RESET (during startup), the EDAC Hard and Soft Interrupts are turned *off*. All RAM cells are subject to the checking and clearing process described earlier in the startup module. Then the EDAC Hard and Soft Interrupts must be cleared. This is accomplished by placing the EDAC Hard/Soft Interrupt Clear signal momentarily *low* (at least one bus cycle). Finally, the EDAC Hard and Soft Interrupts are turned *on*. Note that the EDAC circuitry is always operating, however software can ignore EDAC error signals when necessary. Initial RAM clearing can be performed as word write operations, reducing the time required to perform the initialization process.

b. RAM Wash

The process of performing a RAM wash involves a regularly scheduled software task to read data from RAM. However, the data does not need to be written back. The hardware design automatically writes back a data word that is incorrect (1 bit error, correctable). This process will cause the EDAC circuitry to reset the correction patterns. In the event that a RAM cell develops a bit error, it is desirable that a RAM cell is washed before the RAM cell develops a second bit error. If the wash occurs before the second bit error, the cell and the correction pattern will be updated, reflecting no errors. In the event that a second bit error occurs before the RAM wash, the data will not be correctable and a hard error will be generated. RAM wash can be performed using word write operations. The frequency of RAM washes must occur such that second bit errors do not occur.

As an initial rate of washing RAM, the RAM wash software performs RAM reads on small blocks of continuous data and then returns control to another software routine. This is repeated over and over again until the entire 512 kbytes of RAM have been washed. The process then repeats. A block size of 256 bytes (128 16-bit words) can be washed quickly and simply using a REPS MOVSW which is part of the M80C186 instruction set. There are 4096 separate blocks of 128 bytes within the entire 512 kbytes of RAM. A block wash cycle takes approximately 168 μsec. Interrupts and DMA must be off.

An estimation of the number of SEUs expected in RAM was performed by Oechsel [Ref. 21] based upon data from UoSAT-2, a microsatellite similar to PANSAT regarding its orbit and types of electronics. Conservative assumptions indicated that the number of SEUs expected is 1.0×10^{-6} SEUs/bit/orbit. This equates to an expected time between uncorrectable errors of 1.8 years (nearly the

expected lifetime of PANSAT). In particular, one assumption declares that the RAM wash will occur once

$$\frac{90 \text{ minutes}}{4096 \text{ blocks}} = \frac{1.3184 \text{ sec}}{\text{block}} \qquad (2)$$

per orbit. This rate of washing appears suitable.

This frequency of beginning a block wash is an integral number. If a wash was initiated every second, then the calling of the wash routine would be easy to implement into the clock ISR. This imposes a very small overhead when considering CPU time associated with RAM washing:

c. Processing a Single Bit Error

The interrupt service routine (ISR) for a single bit error consists of removing the interrupt

$$\frac{1 \text{ block}}{1 \text{ sec}} \Rightarrow \frac{168 \ \mu\text{sec}}{1 \text{ sec}}, \text{ which corresponds to an overhead of } 0.0168\%. \qquad (3)$$

condition from the EDAC circuitry by toggling an EDAC control line via the PPI. The EDAC interrupts are level-sensitive. This ISR must acknowledge the interrupt by incrementing a single bit error counter and posting the time and date of the single bit error to the Event Logger. Because EDAC interrupts are level-triggered, further single-bit error interrupt requests will not be detected while software is executing the interrupt service routine. Single-bit errors cause a counter to be incremented, accumulating a count of total single-bit errors.

d. Processing a Dual Bit Error

To reduce the chances of system failure, the dual-bit error interrupt service routine should not be stored in RAM since it is susceptible to dual-bit errors. Because dual-bit errors are not necessarily correctable, software should not attempt to continue operating on the system that experienced the error. The dual-bit error interrupt service routine will place the processor into a halt condition, causing the watchdog timer update to fail and thus shutting down the system controller and starting up the alternate.

E.   MAIN

The convention in a C programming environment is for the function named *main()* to be the first subroutine invoked when the C program is executed. This same convention is followed for the PANSAT ROM Boot software with the exception that after a microprocessor reset, the Startup software executes first, setting up a suitable runtime environment for the actual C program. A flow diagram of the PANSAT ROM Boot software main() routine is shown in Figure 19.

ROM Boot Loader.
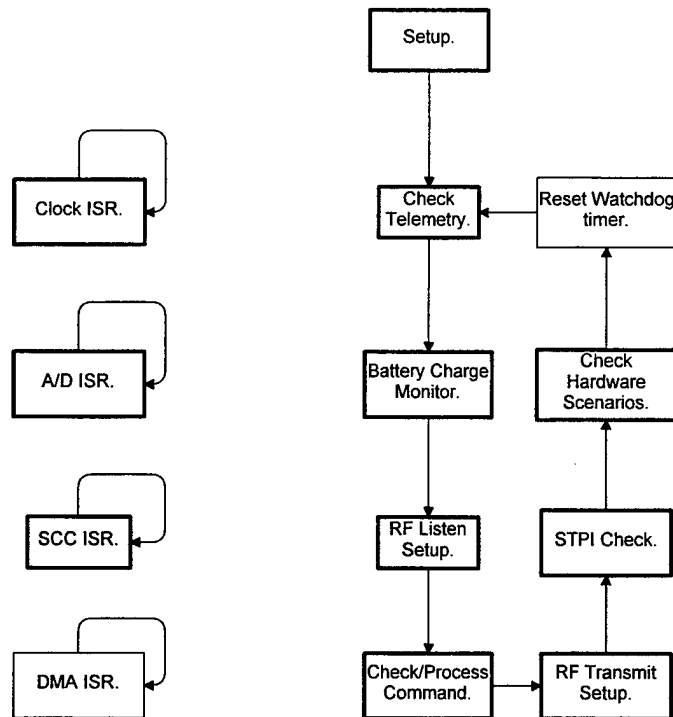
Figure 19. ROM Boot Loader.

This diagram begins with a block named *Setup* which initiates the Interrupt Service Routines, represented on the left side of the diagram. Setup also initializes many variables used to control the Boot Loader loop. The loop calls each of the blocks shown within the loop on a continual basis. Since a multi-tasking operating system is not present within the ROM Boot Software, this software loop depends on each subroutine called to transfer control back to the main loop on a timely basis.

The loop begins by checking telemetry (obtaining the most up-to-date sensor data possible), next is a check on the battery charging, followed by monitoring the RF system for receiving NPS-based transmissions and subsequently transmitting a response. Since this software is also used for testing and integration purposes before spacecraft deployment, the Serial Test Port Interface (STPI) is monitored in case an external computer is sending requests. Next, any incoming command either via the RF system or the STPI is verified and processed. In case there is an apparent hardware failure, alternate hardware configurations are checked and used. And, finally the watchdog timer on board the Electrical Power Subsystem is reset, indicating that the software is functioning and the System Controller that is currently

operating should remain so. The loop then repeats. In the event of successful operation of the Boot loading process, a secondary loader will be uploaded from NPS to the spacecraft. This secondary loader will then take control. The ROM Boot Loader will cease to operate. These higher-level software services are discussed in more detail in the following chapter.

F.   PROGRAMMABLE PERIPHERAL INTERFACE

The Programmable Peripheral Interface (PPI) [Ref. 16] has multiple functions, interfacing with several spacecraft components. The primary use of the PPI is to control the Peripheral Control Bus (PCB) which is discussed later in this section. The PPI also serves to control the EDAC circuitry with three control lines, and performs the Modem mode select using two control lines.

1.   EDAC Control

The PPI has one control line which interacts with the EDAC circuitry. This control line is used to acknowledge to the EDAC that either a hard error or soft error was received and the EDAC state machine should clear the error. Failing to clear the EDAC hard/soft error will result in the continual assertion of that condition. This control line can clear the errors by forcing it low and then high again, (presumably during the interrupt service routine); normally, this control line should remain high.

2.   Peripheral Control Bus

The Peripheral Control Bus (PCB) provides the interconnectivity between all of the functional subsystem electronic components of the spacecraft. This bus identifies each peripheral with a unique address, transmits eight bits of data, and accomplishes this data transfer with read and write control lines.

3.   PPI Control Interface

The active System Controller has control of a PPI by selecting the device associated with PCS1. This PPI is programmed to Mode 2 (strobed bi-directional bus I/O) with a value of 0xC0 to the control port. This allows control of the PCB as well as the three unused control lines for the EDAC and Modem mode selection controls. The configuration of the 8255 and its port assignments are shown in detail in Appendix D.

4.   Peripherals of the Control Bus

The RF system and the EPS on this bus are not duplicated; all other peripherals are paired. Thus, there are two System Control peripherals, containing a System Controller (SCA and SCB), two Analog MUX (AMA and AMB), and two Mass Storage units (MSA and MSB). The least significant bit in the device selection chooses between one of the two ports of the device. Note the distinction of this versus the

sub-address bits (bits 5 and 4); these bits feed through to the selected port, as in the case of addressing a PPI which requires 2 address lines for the PPI registers. The table below identifies each peripheral and indicates the corresponding addresses.

System Name	System [S3 - S0] Address	System Address
RF System	0000, 0001	0, 1
Electrical Power System	1000, 1001	8, 9
System Control A	0010, 0011	2, 3
System Control B	1010, 1011	0xA, 0xB
Analog MUX A	0100, 0101	4, 5
Analog MUX B	1100, 1101	0xC, 0xD
Mass Storage A	0110, 0111	6, 7
Mass Storage B	1110, 1111	0xE, 0xF

Table 8. Peripheral Control Bus Devices.

5.   Reading from the Control Bus

1. Place peripheral select and device sub-address in Port B.
2. Set the Read bit to Low to indicate beginning of read.
3. Toggle the input strobe (STB) to load the data into the input latch.
4. Set the Read bit back to High to indicate end of read cycle.
5. Read the data from Port A.

6.   Writing to the Control Bus

1. Place data in Port A.
2. Place peripheral select and device sub-address in Port B.
3. Toggle the Write bit (High to Low to High) to force the write.

7.   Software Interface

A software interface to allow application tasks to talk to peripherals on the Peripheral Control Bus requires a function to read from a peripheral, and to write to a peripheral. Reading and writing follow a very similar sequence of operations on the control bus.

a. Application Programming Interface

The low level software interface functions, available to a task, are identified below.

Function Name	Description
pcb_portc()	Toggles Port C control bits (0, 1, and 2).
pcb_power()	Toggles power control bits in EPS for each subsystem.
pcb_init()	Initialize the Peripheral Control Bus.
pcb_read()	Reads a peripheral on the control bus and returns the byte.
pcb_write()	Writes a specified byte value to the specified peripheral.

Table 9.   Low Level Peripheral Control Bus Software Interface.

b. Timing Requirements

Reading and writing the PCB are fundamental operations that most software modules use extensively. Thus, it is necessary and useful to determine the amount of time these operations cost. The pcb_write() function takes 180 clocks, and thus at 7.3728 MHz takes 98 μsec to complete. The pcb_read() function takes 220 clocks, requiring 119 μsec. The consequence of the times required are most obvious in the management of the Mass Storage units; this is discussed in a later section.

G.   ELECTRICAL POWER SUBSYSTEM

The Electrical Power Subsystem (EPS) is responsible for generating the power delivered to all other systems within the satellite. Using the Peripheral Control Bus, the switches on the EPS can be toggled. The EPS contains eight ports which contain various switches to control EPS battery functions and to power on/off other devices on the Peripheral Control Bus. Software will keep track (via copies of the EPS registers) of all of the settings for all of the EPS ports. Appendix E contains many tables that describe the EPS ports and the switch assignments of the ports.

1.   EPS Port Organization

Ports 0, 1, 2, and 3 of the EPS are used to control the configuration of the EPS. Port 0 controls the batteries (charge, discharge, and on-line). Port 2 controls the power to the subsystems; this port also contains the current inhibit switch used while reading currents. Ports 1 and 3 control multiplexers on the EPS which are used to select voltage and current measurements. Port 4 is the Watchdog Timer reset switch. Port 5 is used to read back the direction of the current sensors (the only read port via the PCB on the EPS). Ports 6 and 7 are not allocated; however, Port 7 is selected when the Watchdog timer needs toggling (Port 4 written, and then Port 7 accessed in order for Watchdog timer to latch the update request). Ports 0, 1, 2, 3, and 5 are described in detail in Appendix E.

2. EPS Cell Voltage Multiplexing

EPS voltage and current measurement signals are all multiplexed onto one analog signal and delivered to the LM12458 A/D converter on-board the System Controller. The following section describes the addressing needed to perform EPS voltage and current measurement selections.

a. Low Battery Cell Voltage Selections

The Low Battery Cell Voltage MUX is used to select certain battery cells for voltage measurements associated directly with the Low Cells (0 and 1). This MUX is also used to index into the Medium and High Cell Voltages MUXes. To select battery cells 0A, 1A, 0B, 1B for voltage measurements, only Port 3 needs to be used. Since Port 3 is used to index into the Medium Cell Voltage MUX, these control bits are zeroed when only selecting cells 0 and 1.

b. Medium Battery Cell Voltage Selections

The Medium Battery Cell Voltage MUX requires manipulating the Low Cell Voltage MUX as well as the Medium Cell Voltage MUX. Prior to selecting the Medium Battery Cell Voltage, the Low Battery Cell Voltage MUX must be set correctly. Since the control of the Medium and Low Battery Voltage MUXes are controlled both with Port 3 of the EPS, all of the selection actions can take place in one write to Port 3. The four least significant bits (1111) set up the Low Cell MUX to allow Medium Cell MUX selections to pass through.

c. High Battery Cell Voltage Selections

The High Battery Cell Voltage MUX selection method depends on the cell. Cells 5A/B, 6A/B, and 7A/B require the following selection method. First, select the High MUX input through the Low Cell MUX (Port 3 = 0000 1000). Furthermore, the Current Select must be kept disabled (Port 1: bit 0 = 0). Then, via Port 2, make a selection. Cells 8A/B and 9A/B require the following selection method. First, select the High MUX input through the Low Cell MUX (Port 3 = 0000 1000). Furthermore, the Current Select must be kept disabled (Port 1: bit 0 = 0).

3. EPS Cell Voltage and Current Multiplexing

The eleven current sensors are used to read roll rate experiment currents and also the solar panel and battery currents. The Low Cell Voltage MUX selectors are used to index voltage measurement, as well as the current selectors. Prior to current selections, Port 3 should be set to allow current selections to be made; this is accomplished by setting Port 3 = 0000 1100 (for roll rate), Port 3 = 0000 1010 for batteries and solar panel bus. Then the current selection is made. After the address selections, the Spacecraft Power Current Inhibit control must be disabled (i.e. enabled) by setting Port 1, Bit 0 to 1. Then, the Spacecraft Power Current Strobe must be strobed from high to low to high (Port 1, Bit 1). Then a reading can be

made. Following the A/D reading, the Spacecraft Power Current Inhibit must be set again (Port 1, Bit 0 set to 0). Appendix E shows the configurations needed for current measurements.

H.   SERIAL COMMUNICATIONS

1.   Modem Control

A System Controller operates the Modem using I/O commands issued from the CPU. The Modem has a set of registers within the PA-100 [Ref. 17] and a separate 8-bit latch. Furthermore, a SC has the ability to turn on and off the entire Modem board via the PPI control port C bit 0. Modem setup and control requires a set of registers within the PA-100 to be configured in a certain sequence. Furthermore, feedback from the PA-100 is also required. In order to simplify the programming interface to control and monitor the Modem board, two functions exist to assist high-level layers of software. A data structure can be created which contains a pair of data, an *address* and a *value* associated with that address. The *address* corresponds either to a register within the PA-100 or the 8-bit latch, and the *value* corresponds to the data to be placed at that *address*. Thus, tables are sent to the Modem board when a new configuration is desired.

2.   RF Control

The RF unit [Ref. 36] is controlled via the PCB using one 8-bit latch. Logic within the RF unit along with settings made by a System Controller control the selection of a low noise amplifier (LNA), a high pass amplifier (HPA) and a corresponding power level, a local oscillator (LO), the power applied to the selected LNA and also HPA, and control of transmit and receive switches. A System Controller has primary and alternate components (LNA, HPA, LO) within the RF unit. The RF unit takes a System Controller's preferences via PCB commands (Table 10) as well as which SC is active and enables the correct components.

RF PCB Control Port Bit	Control Bit Name	Description
7	HPA	0 = Selected HPA off, 1 = Selected HPA on
6	LNA	0 = Selected LNA on, 1 = Selected LNA off
5	P1	Power level control (most significant bit)
4	P0	Power level control (least significant bit)
3	LHP/LHA	0 = Primary LNA/HPA, 1 = Alternate LNA/HPA
2	LOP/LOA	0 = Primary Local Oscillator, 1 = Alternate Local Oscillator
1	Tx/Rx	0 = Transmit, 1 = Receive (Signal Path Relay)
0	T/R	0 = Transmit, 1 = Receive (Antenna Relay)

Table 10. RF Unit Control Port.

API function calls are provided for ease of programming as well as making the source code very explicit to understand regarding the references to the RF unit. The functions are shown in Table 11.

Function Name	Description
rf_power()	Toggles power on and off to the RF unit.
rf_set()	Sets a particular RF control bit via the PCB.
rf_timer()	Starts the Transmitter timer using Timers 0 & 1 of the CPU.
rf_txpower()	Sets transmit power using the 2-bit attenuator on the RF unit.

Table 11. RF API function calls.

3.   SCC Drivers

One master interrupt service routine (ISR) handles the input and output services of both Channel A (synchronous) and Channel B (asynchronous) of the Serial Communications Controller [Ref. 28]. Channel A I/O is handled with DMA, except the special conditions which cause an interrupt. However, all I/O for Channel B is handled within this service routine. Since the SCC has only one interrupt source into the CPU, all SCC interrupt conditions invoke this one ISR. All conditions are checked within this ISR and are handled on a priority basis. The condition checking repeats within the ISR until all SCC interrupt conditions have been serviced. An outline of the master SCC ISR is shown in Figure 20.
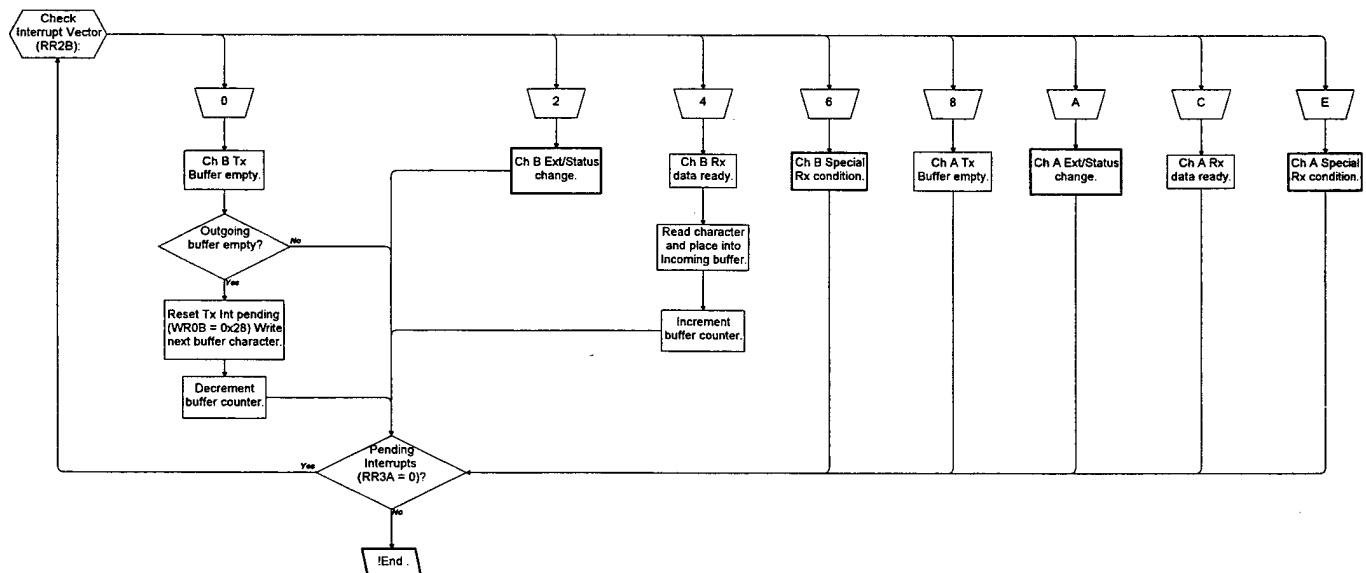
Figure 20. SCC Interrupt Service Routine Structure.

53

a. Asynchronous Services

A variety of software routines allow control of Channel B of the SCC, providing an easy yet sophisticated interface with the asynchronous communications port. The core drivers that allow input and output to Channel B are serial_out() and serial_in(). These functions simply insert or remove characters into buffers that are handled by the interrupt service routine. At a higher-level, software routines allow command line input and C Standard Library like printf() capabilities for complex display output.

b. Synchronous Services

Synchronous data is received and transmitted in blocks as packets. All higher-level software modules ultimately want to send a packet of data or receive a packet of data. Since all synchronous I/O is handled with the DMA between the memory and SCC, packet send and receive requests require an initial DMA setup, and some follow up after the transaction. Since the satellite only works in a half-duplex mode, assumptions regarding the state of transmit and receive are made and simplify the design. When not transmitting, the SC places the SCC into a receive mode with the DMA already set up to transfer incoming data bytes. Thus, during this time, it is not necessary to set up anything regarding the transmission of data. When data is required to be sent out a packet is placed into a buffer and remains there until the receiver is finished with packet reception. Then, the transmitter is set up, including the DMA. During this time, it is not necessary to set up anything regarding the reception of data.

Synchronous services are complicated by the fact that besides the SCC, communications through Channel A also rely on the PA-100 modem and the RF unit. Thus, the PA-100 and RF unit require some monitoring in case there is a failure. Failure may only be due to a functional mode of the hardware and not circuit failure. Thus, these units may need to be reset or programmed with different parameters. Furthermore, extended failure may require a different hardware configuration to be chosen.

I. TELEMETRY

1. Scheduling of the LM12H458

Telemetry from sensors (voltages, currents, and temperatures) is routed onto the inputs of the A/D converter on the System Controller, the LM12H458 [Ref. 24]. An interrupt service routine (ISR) is responsible for completion of data conversion, and scheduling of another acquisition under the automated function of the A/D converter. Since the LM12H458 can be given a program which is a sequence of data acquisition steps which can be run independently of the CPU, the A/D converter is programmed to make several acquisitions across multiple input sensors without intervention of the CPU.

In order to simplify the software responsible for operating the A/D converter, a schedule was created that indicates for any sequence in the data acquisition, which sensor signals are to be accessed and converted. The entire data acquisition cycle was broken up into 42 periods, numbered 0 through 41. After the 41th period, all sensor data has been read and converted at least once. Since some data points need more frequent readings than others, certain sensors are read multiple times during the 42 periods. The sensors from the EPS require multiple readings per period. The CPU must store the converted values at the end of each Period and prepare the LM12H458 for the next Period before resuming other activities. Appendix F shows the schedule of A/D conversions.

2.   LM12H458 Setup and Interrupt Service Routines

Three subroutines implement the software necessary for data acquisition. The first is an initialization routine which programs the LM12H458 into the desired mode, calibrates it, programs the first sequence of acquisition instructions, and finishes leaving the LM12H458 running independently.

Another, the core of the acquisition, occurs within the Interrupt Service Routine for the LM12H458, diagrammed in Figure 21. At the end of each data acquisition Period, which can have up to 5 converted samples or as few as one, this routine is invoked via a hardware interrupt. If necessary, current directions are read for the just completed acquisition. If temperature sensors are used for the following acquisition period, the TMUXes are programmed. Furthermore, the EPS is programmed to multiplex the soon to be read sensor. Finally, the LM12H458 instructions are programmed and the service routine is finished. After the completion of all 49 Periods, this ISR also sets a flag to notify other software tasks that a complete set of new data has been converted. Also, at the completion of the last Period, the LM12H458 is instructed to complete a calibration, and thus control of the CPU is returned to other software tasks. The completion of the calibration causes another ISR to program the LM12H458 to begin again.

55

Main ("Pause") ISR
for A/D (LM12H458)
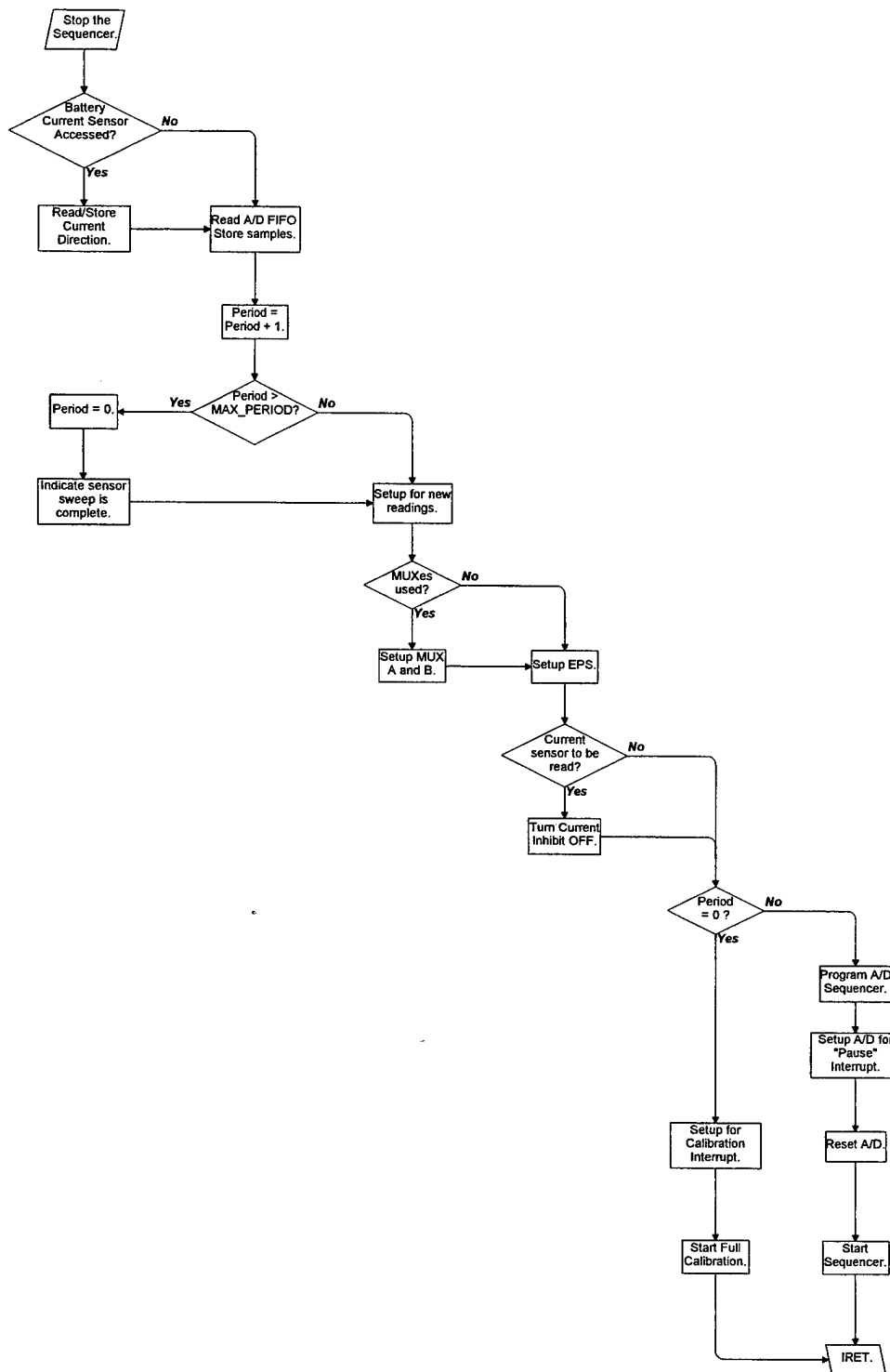
Figure 21. A/D Main ISR.

3. Data Gathering - Temperature Multiplexers

All temperature signals sensed using thermistors (all temperature measurements except two IC temperature probes on the System Controller and Modem boards) are multiplexed onto the Temperature Analog Multiplexing units (TMUX) [Ref. 37]. These signals pass through an appropriate signal conditioning circuit and are then available to the LM12H1458 for analog-to-digital conversion. This conditioned analog signal is passed to the LM12H458, on-board a System Controller. All of the TMUX selections are controlled using the Peripheral Control Bus.

TMUX control is handled by four address lines and four selector lines using the Peripheral Control Bus. All multiplexers are fed the same address lines. However, four separate selection lines provide the TMUX enabling to each individual TMUX. The following table indicates the assignments of the bits that describe the TMUX configuration.

Bits	Function
7	Unused
6	Unused
5	MUX 1 Select (subaddress) *Channels 16-31*
4	MUX 0 Select (subaddress) *Channels 0-15*
3 - 0	MUX Address (applies to each MUX)

Table 12. TMUX Control.

Temperature sensors are named TS*x*, where x is a number beginning with 0. Temperature sensors which deliver a signal to TMUXA are even numbers (including the beginning 0), and temperature sensors to TMUXB are odd numbers. Furthermore, all sensors are redundant, for every sensor providing a signal to TMUXA, another sensor exists to provide the same sensor location to TMUXB. The number system is such that if x is a sensor for TMUXA, *x + 1* is the redundant sensor for TMUXB.

4. Data Conversion

Data conversion for voltages, currents, and temperatures are explained in this section. Note that N is an unsigned value from the A/D converter. And where appropriate, *SIGN* is either +1 or -1 and determined from reading the current direction sensor.

a. Battery Current Conversion

The current going into (positive) or leaving (negative) a battery is determined using Equation 4.

$$I = 2*SIGN*\left[N*\left(\frac{5}{4095}\right) - 2.5\right] = SIGN*(N*0.002442 - 5) \qquad (4)$$

b. Spacecraft Bus Current Conversion

The calculation of the spacecraft bus current is identical to the conversion of the battery current except that there is no direction, and thus the use of *SIGN* is eliminated, Equation 5.

$$I = 2*\left[N*\left(\frac{5}{4095}\right) - 2.5 \right] = N*0.002442 - 5 \qquad (5)$$

c. Battery Voltage Conversions

Battery voltage measurements are accumulated voltage readings across cells in series. Starting with cell 0 as the first cell in series, each successive cell has a voltage measurement across the entire series, and not the individual cell. Accumulated Cell Voltages require a conversion weight, W (Table 13), which depends on the cell.

Cell	Battery A	Battery B
0	1.0	1.0
1	1.0	1.0
2	1.6953	1.6969
3	1.6997	1.7015
4	1.7014	1.7029
5	2.4267	2.4283
6	2.4262	2.4314
7	2.4311	2.4317
8	4.9	4.93
	Factor (W)	Factor (W)

Table 13. Battery Voltage Conversions.

First, the accumulated cell voltages are converted (Equation 6). These accumulated cell voltages are the steps of voltages measured across the entire cell series, starting with cell 0 as the base cell with direct reference to ground.

$$V_a = N*\left(\frac{5}{4095}\right)*W = 0.001221*N*W \qquad (6)$$

Next, each individual cell voltage can be calculated by subtracting the cell's accumulated voltage from the cell previous to it, with exception of the first cell, cell 0. This is shown in Equation 7.

$$V(i) = V_a(i) - V_a(i-1), \qquad 1 \le i \le 8$$
$$V(0) = V_a(0) \qquad (7)$$

58

d. Thermistors

An Omega 440048 thermistor's [Ref. 38, p. D-4] temperature value is converted using a table lookup. This is because the conversion is based upon the mathematically intensive Equation 8. The table lookup performs a binary search where there are a maximum of seven compares needed for a lookup. The table is used by taking the value from the A/D converter and finding the closest match in the table. The position of the match in the table indicates the temperature of the sensor. Appendix G contains a complete discussion of the conversion process.

$$T = \left[\frac{1}{A + B * \ln(R) + C * \left[\ln(R) \right]^3} \right] - 273.15$$

$$R = \frac{V}{I_C}$$

where R is the resistance of the thermistor, $\qquad$ (8)

V is the voltage sensed across the thermistor,

and I_C is the Calibration Current. A, B, and C are coefficients

choosen to best fit temperature values in the range from $- 0^{\circ}$ C to 30° C.

$A = 9.306 \times 10^{-4}, B = 2.218 \times 10^{-4}, C = 1.253 \times 10^{-7}$.

e. Temperature Sensors (ICs)

Conversion of the IC temperature sensors is straight-forward as shown by the relationship expressed in Equation 9.

$$T = (N - 0.5) * 100 \quad (9)$$

5.   Data Recording

At regular intervals, preset to every minute but modifiable by command from the ground station, all of the sensor data and various software statistics are saved to the mass storage devices. The purpose of recording this data is two-fold. First, the system software uses this recorded data to make decisions, in particular when the satellite has a power reset, or a System Controller is turned off and its alternate turned on. When powered on, a System Controller attempts to find previously saved data in the mass storage which will describe the prior state of the satellite. In the case of the first time the satellite operates (i.e. after launch), the mass storage is completely initialized. The Battery Charge Monitor (discussed in the next chapter) relies on this recorded data to make intelligent decisions regarding the state of the batteries if

possible. The data is recorded so that it is accessible to the ground station for detailed analysis. An overview of the telemetry record contents is shown below in Table 14. A detailed description of the telemetry record is found in Appendix J, documented in the software source code.

Item	Quantity	Description
Time/Date	1	Spacecraft time and date recorded in number of ticks (60 Hz).
Temperatures	37	Temperatures of modules, batteries, and solar panels.
Voltages	19	Voltages of battery cells and spacecraft bus.
Currents	11	Currents of batteries, spacecraft bus, solar panels.
Hardware Configuration	37	Subsystem port settings.
BCM	24	Battery Control Monitor parameters.
EDAC Errors	1	Number of EDAC soft errors.
SU Fails	1	Number of superuser access attemps that failed.

Table 14. Telemetry Record Contents.

J.   MASS STORAGE INTERFACE

A mass storage unit (MSU) provides four megabytes of Static RAM (SRAM) to serve as file storage for the Spacecraft Software. This unit is intended to save user messages (mailbox storage) and archived telemetry. In addition, half a megabyte of Flash storage is available for duplicating telemetry.

1.   Hardware Interface Via the PCB

The MSU is a data storage device of both volatile and non-volatile solid state memory devices. The data address to be accessed is presented and the appropriate control (read or write) is indicated. Four and a half megabytes of random access storage are incorporated within the Mass Storage Unit. The Peripheral Control Bus is used to interface the Mass Storage Unit and the System Controller. A PPI is located at base address of the Mass Storage PCB address (see Table 15). Twenty-two bits of the PPI are used to latch an address. There is one read signal and one write signal, and one additional signal used to indicate if SRAM or Flash is to be accessed. Table 16 indicates addressing and control usage within the PPI.

PCB S3-S0	PCB A1-A0	Usage for Mass Storage Units
		Mass Storage A
6	0	PPI Port A
6	1	PPI Port B
6	2	PPI Port C
6	3	PPI Control Register
7	0	Data (read and write)
		Mass Storage B
E	0	PPI Port A
E	1	PPI Port B
E	2	PPI Port C
E	3	PPI Control Register
F	0	Data (read and write)

Table 15. Mass Storage Unit: PCB Interface.

PPI Port	Bits Used	Usage for Addressing or Control
Port A	D0 - D7	Memory Addresses (A0 - A7)
Port B	D0 - D7	Memory Addresses (A8 - A15)
Port C	D0 - D5	Memory Addresses (A16 - A21)
Port C	D6	Select: 1 = Flash, 0 = SRAM
Port C	D7	*Unused*

Table 16. Mass Storage Unit: PPI Interface.

For initialization, the PPI should be programmed as an output only device (Ports A, B, and C all 8-bit outputs); this is accomplished by writing a 0x80 to the PPI Control register. Port C controls both the selection of either the Flash or the SRAM devices and the most-significant address bits of both device types. Since selected Flash devices draw less current than selected SRAM devices it is best to select the Flash devices when a Mass Storage board is powered on (but not being read or written to). Furthermore, A18 and A19 select only high address bits of the SRAM; selecting a Flash device in this address range causes a non-existent Flash device to be selected, drawing even less power. Therefore, a powered-on Mass Storage board, when not performing reading or writing, should have its Port C set to 0x48.

2.   Software Interface

Device driver software allows the functions of formatting (clearing memory in preparation of writing data), writing to, and reading from the Mass Storage units. Although the entire memory space of one mass storage can be addressed as a continuous address space, due to logical use of the memory, the memory space is visualized and thus segmented into two memory types: the volatile 4 Mbyte SRAM, and the non-volatile ½ Mbyte Flash.

61

a. Reading and Writing Requirements

Reading from either type of memory is straight-forward. An address is programmed into the PPI on the Mass Storage board, and then a PCB read retrieves the value of the cell pointed to. Writing to the SRAM is similar to reading, except a PCB write is issued. However, writing (and erasing) the Flash involves writing specific address and data sequences into the command register of the Flash device [Ref. 39] defines these register command sequences which allow writing, erasing, and checking manufacturer and device type codes.

Command Sequence	First Bus Write		Second Bus Write		Third Bus Write		Fourth Bus Write		Fifth Bus Write		Sixth Bus Write	
All values are in Hex.	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read/Reset	5555	AA	2AAA	55	5555	F0	RA	RD	*RA = Read Address, RD = Read Data*			
Scan	5555	AA	2AAA	55	5555	90	0/1	Code				
Write	5555	AA	2AAA	55	5555	A0	WA	WD	*WA = Write Address, WD = Write Data*			
Erase	5555	AA	2AAA	55	5555	80	5555	AA	2AAA	55	5555	10

Table 17. Mass Storage Unit: Flash Commands.

b. API Functions

API function calls for each memory type are provided for ease of programming as well as making the source code very explicit to understand regarding the references to the Mass Storage memories. These functions are shown in Table 18. For redundancy purposes, the ROM Boot Loader software writes all recorded data to each Mass Storage unit and to both memory types. Thus, the data can be quadruplicated. All records of data written to a Mass Storage device use a Cyclic Redundancy Check (CRC) [Ref. 40] for read-back verification. The CCITT CRC-16 will detect all single-bit, dual-bit, odd numbered, and bursts of fewer than 17 bits wide types of errors. Detection of other errors is about 99.997%. This is the same CRC algorithm as used within the SCC except that the seed CRC is 0 (instead of 0xFFFF).

Function Name	Description
msu_erase_flash()	Erase entire Flash (sets cells to 0xFF).
msu_read_flash()	Read specified number of bytes from a specified address in Flash.
msu_write_flash()	Write a specified number of bytes from a specified address in Flash.
msu_scan_flash()	Scan Flash devices for manufacturer and device type codes.
msa_set_addr()	Set the mass storage address pointer to a specified location.
msu_erase_sram()	Erase entire SRAM (sets cells to 0xFF).
msu_read_sram()	Read specified number of bytes from a specified address in SRAM.
msu_write_sram()	Write a specified number of bytes from a specified address in SRAM.

Table 18. Mass Storage Unit: API Functions.

c. Timing Requirements

Reading and writing to a Mass Storage unit takes a considerable amount of time especially writing to the Flash memories. Therefore, it is critical that the amount of time required to accomplish such operations is determined and taken into consideration when designing and writing the device drivers as well as the high-level code that uses the Mass Storage. In general, reading and writing to the SRAM, and reading from the Flash take a similar amount of time. However, writing to Flash is nearly eight times slower. Since data may be read and written in blocks, a general formula was determined which gives a per byte time requirement. Reading and writing data in blocks is preferred (rather than single byte function calls) because the overhead of multiple function calls is reduced, and address incrementing techniques can be exploited (e.g. only updating the address digits that change). Table 19 shows the formula and the transfer time for 256 byte blocks.

Function Name	Clock Cycles Required Per Byte	Time (n = 256)
msu_read_flash()	Clocks = 1210 + n*(862) + (n-1)*(31) + 40	31 msec
msu_write_flash()	Clocks = 320 + n*(7668)	266 msec
msu_read_sram()	Clocks = 1207 + n*(862) + (n-1)*(31) + 40	31 msec
msu_write_sram()	Clocks = 1207 + n*(834) + (n-1)*(31) + 40	30 msec

Table 19. Mass Storage Unit: Timing Of Operations.

This concludes the presentation of the System Controller software device drivers. These drivers are necessary to provide simple and direct control of the hardware peripherals (electronics of the System Controller as well as the electronic modules of the spacecraft). However, the goal of the software developed for PANSAT is to provide a system which orchestrates all of the hardware of the spacecraft. This software is referred to as the System Controller high-level software and is presented in the next chapter.

VI. SYSTEM CONTROLLER HIGH-LEVEL SOFTWARE

A. DESCRIPTION

Upon completion of the hardware initialization and the creation of a runtime environment suitable for higher-level layers of software, software progresses to the central loop contained in the module *main.c*. Operation then consists of monitoring and charging the batteries while establishing primitive communication with the PANSAT Ground Station at NPS to begin the upload of the Kernel and the PHT loader. Thereafter, a higher level protocol will upload BAX, the primitive Telemetry collector, and an AX.25-aware loader. Then, using AX.25, the FTL0 protocol, the File System, and Bulletin Board services software are uploaded. Finally, the spacecraft is ready for general use.

B. SERIAL COMMUNICATIONS - Serial Test Port Interface

The Serial Test Port Interface (STPI) is an asynchronous serial communication interface that exchanges messages between the System Controller and an external computer acting as a Data Terminal Equipment (DTE). The DTE is a dumb-terminal which is capable of sending data and displaying messages sent from the SC. A large set of commands is implemented to operate and test the spacecraft via the STPI. The command menu is shown below in Table 20. Responses from the commands (sent by the SC) vary depending on the command; however, all responses print out as readable messages on the terminal.

Command	Description	Param. 0	Param. 1	Param. 2	Param. 3	Param. 4
in	Input byte	port-16				
inw	Input word	port-16				
out	Output byte	port-16	data-8			
outw	Output word	port-16	data-16			
peek	Read byte	segment-6	offset-16			
peekw	Read word	segment-16	offset-16			
poke	Write byte	segment-16	offset-16	data-8		
pokew	Write word	segment-16	offset-16	data-16		
dump	Dump paragraph	segment-16	offset-16			
pcbr	PCB read	select	subaddr			
pcbw	PCB write	select	subaddr	data-8		
ad	A/D ISR control	on/off				
tlm	Get recent telemetry					
time: g	Get time					
time: s	Set time	date-time				
bcm: p	BCM: on/off	on/off				
bcm: c	BCM: configure	config data...				
bcm: s	BCM: status					
eps:p	EPS: power control	device	on/off			
eps:b	EPS: battery control	battery	switch	on/off		
eps:v	EPS: read voltage	select	cell #			
eps:i	EPS: read current	select	sp #			
msu:p	MSU: power	device	on/off			
msu:e	MSU: erase flash	device	sector#			
msu:r	MSU: read	device	type	address	length	
msu:w	MSU: write	device ,	type	address	length	data....
tmux: p	TMUX: power	device	on/off			
tmux: r	TMUX: read	device	channel #			
modem:p	Modem: power	on/off				
modem:m	Modem: mode	clear/spread/test				
modem:r	Modem: read status					
modem:w	Modem: write	data...				
rf:p	RF: power control	on/off				
rf:c	RF: configure	config data...				
scc: i	SCC: initialize					
scc: r	SCC: read register	register				
scc: w	SCC: write register	register	data-8			
scc: h	SCC: hunt					
scc: rx	SCC: Receive					
scc: tx	SCC: Transmit	length	data...			

Table 20. STPI Commands.

66

C.   BATTERY CHARGE MONITOR

Two battery packs are within the spacecraft [Ref. 41]. Each battery pack consists of nine nickel-cadmium cells with a standard capacity of approximately 4.4 AmpHours. The batteries are for eclipse power and power regulation and conditioning circuitry while in sun-soak. The power system must be able to switch from external solar power to internal battery power without major power spikes or fluctuations. The EPS relies on the System Controller to activate switches and to determine charge levels of the batteries. Both batteries will be depleted beyond the capability of operation at launch due to the storage time between integration and ejection from the Shuttle. A trickle charge circuit provides battery charging at the beginning-of-life while the spacecraft operates in the sunlight. This will allow a low power mode of operation during eclipse in the very early stage of the mission until the batteries are sufficiently charged. During spacecraft operation the battery control algorithm will use temperature, current, and voltage measurements to determine the state-of-health of the two batteries. The state-of-health determines which battery is on line (providing buffered power to the spacecraft bus during a sun-soak and full power while in eclipse), and which battery is being charged.

The battery charge monitor (BCM) is an algorithm that controls the charge and discharge cycles of both batteries. It controls and monitors the charge and discharge cycles using overcharge times, cell temperature profiles, cell voltages and charge/discharge currents. Depending on the battery status, a *Target* battery is selected to be charged. Also, once the satellite is launched into space, the power provided from the solar panels must be distributed and regulated. While the satellite is in eclipse, power from the batteries is necessary to keep PANSAT operational. The battery has to provide additional power during sunlight in the event that the solar panel output is insufficient, such as during transmission. The *Online* battery provides this power. Furthermore, the algorithm indicates the available power for operations, depending on the charge state of the batteries: low, stand-by, or normal power operation modes. According to the mode, certain subsystems will be turned off to guarantee operation.

The BCM also checks for corrupt data which is stored to review the charge state history of the batteries. When such an error occurs, the algorithm will set the satellite to a default condition and restart the battery charging based on measurements only, beginning new charge state history data. The algorithm also detects either sunlight or eclipse. This is necessary to activate certain switch configurations to allow continuous operation when the satellite is going through a transition from sunlight to eclipse or vice versa.

1.   BCM Top Level

The BCM is essentially a large series of questions which are asked on a continual and frequent basis. The questions consider recent measurements (temperature, current, and voltage) of the batteries, past measurements, and whether or not the spacecraft is in eclipse. This series of questions determines how the switches within the EPS are turned on and off in order to maintain correct and efficient use of the batteries.

These switches control which battery is on line, charging (trickle and normal), and discharging. A top-down view of the algorithm is shown, starting with the main structure shown in Figure 22.

Figure 22. Battery Control Algorithm Top Level.

68
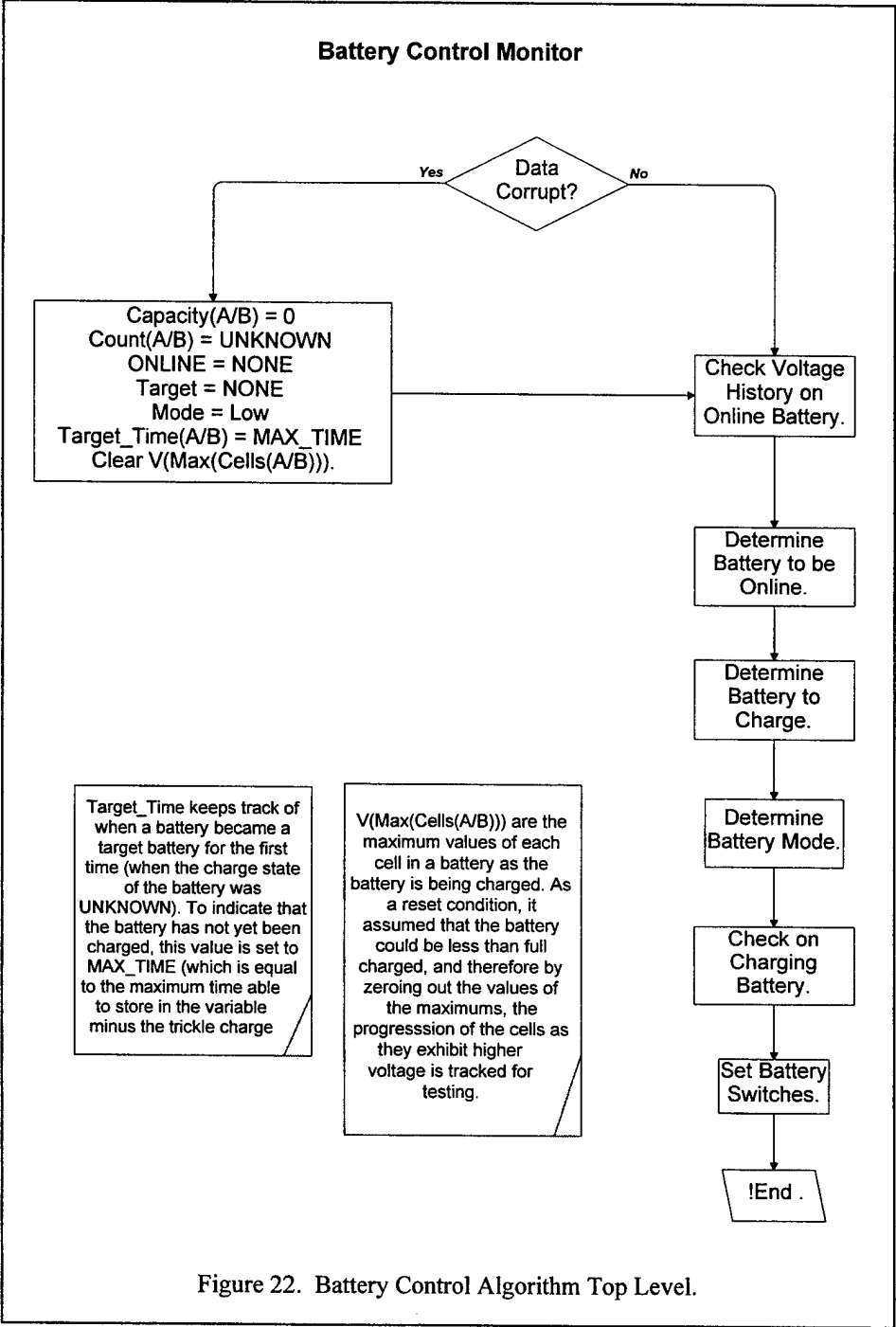
A handful of variables describe the state of the BCM, such as Capacity, Count, Online, Target, Mode, and cell voltage maximums. Capacity indicates how much charge is presently in a battery and is expressed in AmpHours. The default Capacity for each battery is 0 AmpHours (empty). Count indicates how many times a battery has been recharged as opposed to overcharge (discussed later). Initially, a value of -2 (UNKNOWN) is assigned which means the battery needs immediate overcharging. Target indicates which battery is currently being charged. To begin, the Target is None (neither battery since it must be decided). Mode depends on the charge state of the batteries. Initially, Mode is Low (meaning satellite operations should consume the least amount of power), and progresses to Standby, and Normal. Cell Voltage maximums keep track of the highest voltage each cell in a battery reaches during a charge cycle (while it is the Target).

Other useful parameters are depth of discharge (DOD), temperature references, an overcharge voltage threshold, and timers. A previously charged battery which is online and losing charge capacity reaches its depth of discharge (DOD) when it has 60% of its full capacity. This is a condition that should trigger the battery to no longer be used for discharging and to begin a new charge cycle as soon as possible. Temperature references note the temperature of batteries when a battery is first placed on line or started charging. Such temperature references are used to monitor rapid temperature increases which can signify a battery reaching overcharge; thus, this is a safety mechanism. An overcharge voltage threshold is a voltage that is temperature dependent and indicates when a battery is approaching an overcharge condition. This condition triggers the starting of timers which further monitor the amount of charge a target battery receives before completing a charge cycle.

2.   Battery Use Eligibility and Preference

As a necessary requirement for a battery to be eligible to be Online or the Target, the conditions of the batteries are examined as shown in Figure 23. There are three topics of interest which can persuade whether or not a battery is suitable for use. First, if a battery is already being charged (is the Target) and it has a cell voltage below 0.9 V, then that battery should not be used. It should be considered defective. Second (and this applies potentially to both batteries), if a battery is not currently being charged and has already been charged then its cells must have a minimum cell voltage of 1.1 V. Otherwise, that battery should not be used. It should be considered defective. If both batteries are still eligible for operation, then the temperatures of the batteries are examined (Table 21); that is, both batteries are still eligible for use. If the temperatures (an average temperature of all cells) of both batteries are between 0°C and 35°C, then both batteries are eligible for use and there is no preference. Any preference given will be a function of the charge state of the batteries (discussed later). However, if the temperatures are out of the range just described, then one of the batteries will be preferred over the other.
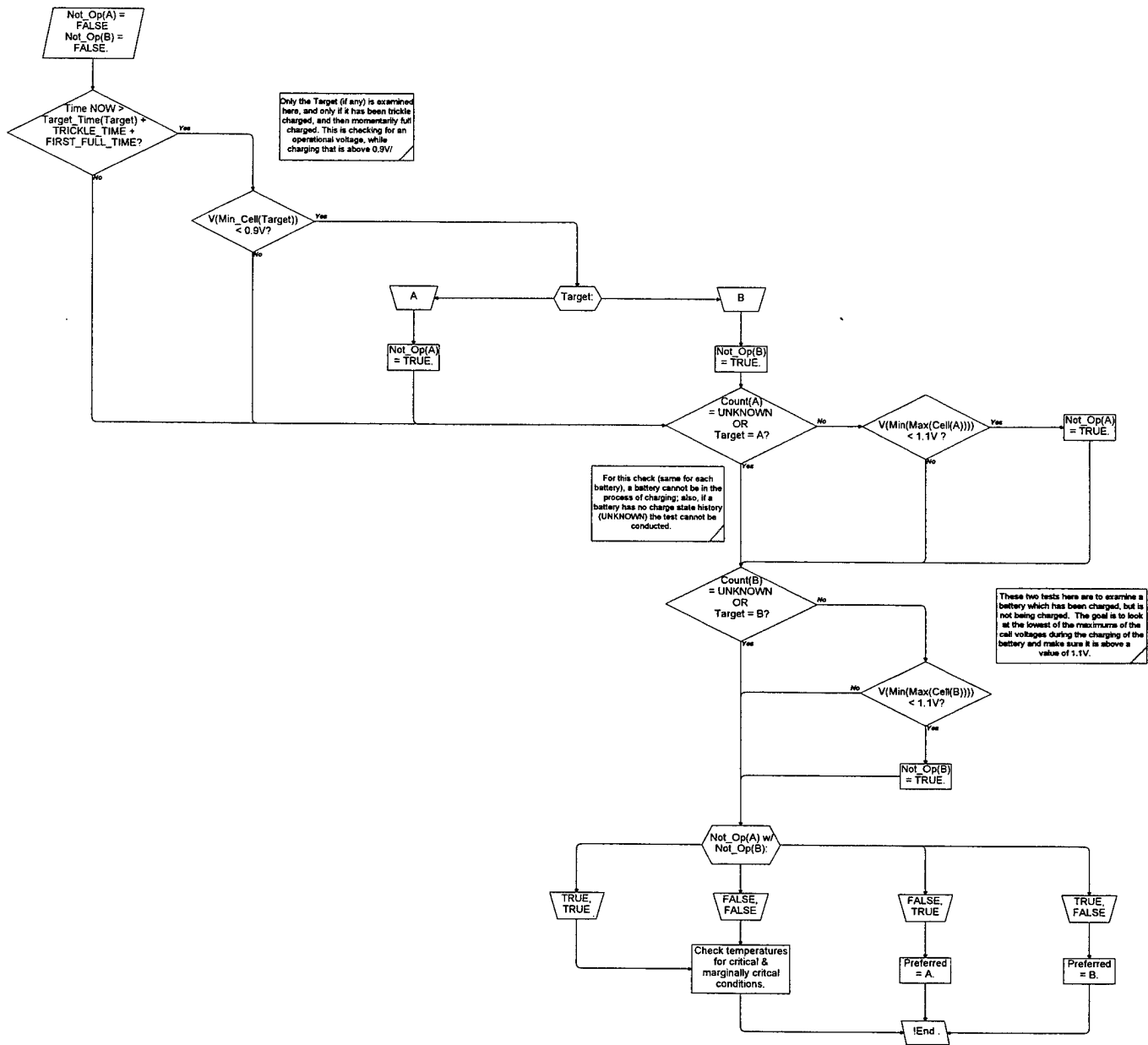
Figure 23. Battery Preference.

70

Search begins by using lowest temperatures in this table and working towards higher temperatures.

Battery A

	< -10	<= 0	< 35	<= 45	> 45
<-10	Use Warmer	A	A	A	B
<= 0	B	Use Warmer	A	B	B
< 35	B	B	No Preference	B	B
<= 45	B	A	A	Use Cooler	B
> 45	A	A	A	A	Use Cooler

Battery B labels the left column.

Table 21. Battery Preference Based On Temperatures.

3.   Determine Online Battery

In order to determine which of the batteries should be online, many questions are answered (Figure 24). First, a battery preference (as described above) is considered. If no such preference exists, then the following is considered. If there is not already an online battery then the battery with the most capacity (stored charge) is chosen. Otherwise, if a battery is already designated as online, it remains online as long as its voltage has not dropped below a voltage threshold of 1.1 V and its capacity is above its DOD. If the capacity of the online battery is below its DOD and if the other battery is not being charged, then the other battery becomes the online battery allowing the depleted battery to begin a new charge cycle. However, if the online battery must remain online because the other battery is being charged (disrupting charge cycles is not preferred) then this condition is flagged and power consumption within the spacecraft must be reduced.
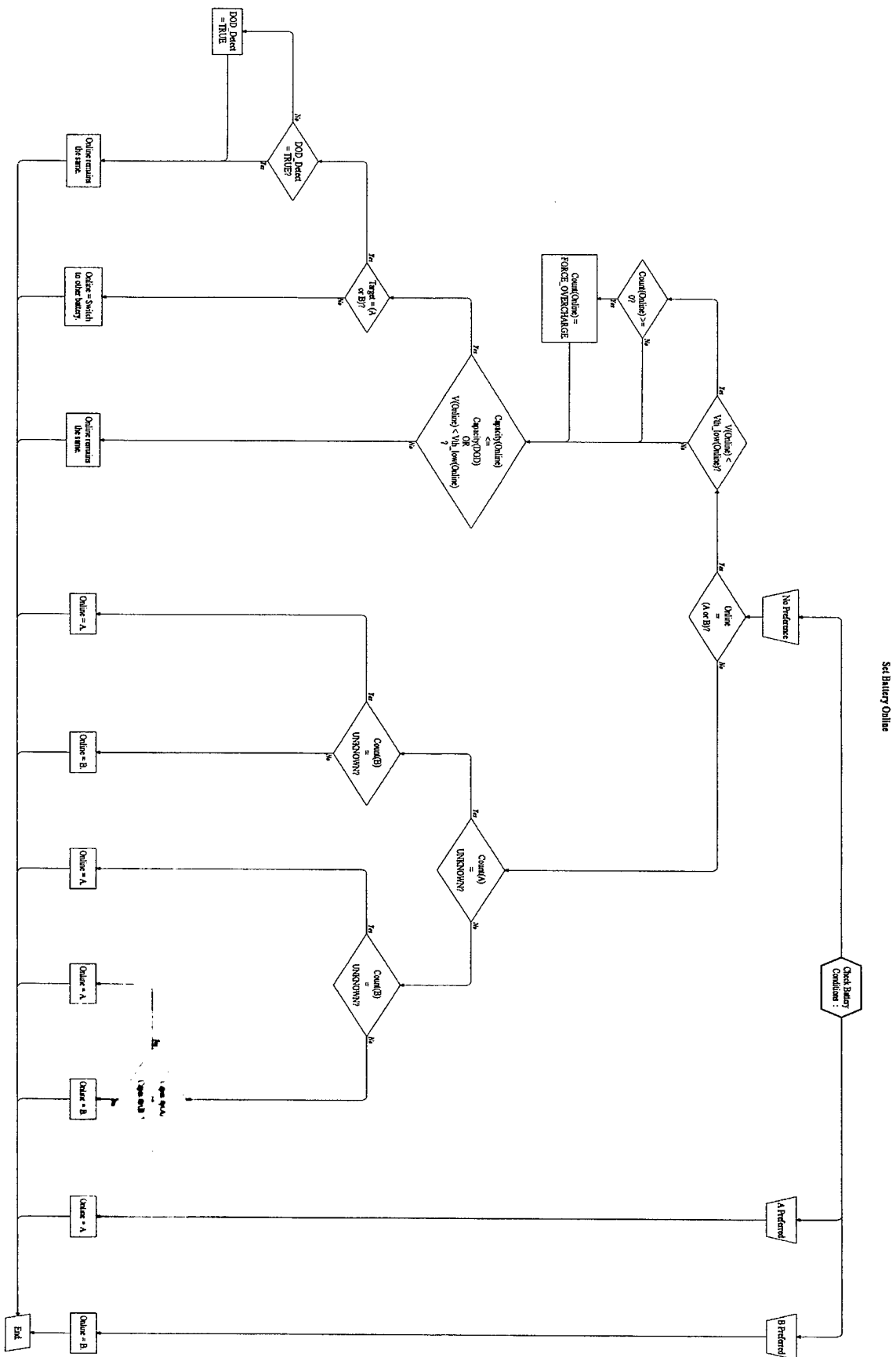
Figure 24. Battery Online.

4.   Determine Target Battery

During the next portion of questioning (Figure 25), if applicable, a battery is chosen to become the Target and will begin being charged. Again battery preferences take precedence in the decision as to which battery should be the Target. If no such preference exists, and there is not already a Target, then the battery that is below its depth of discharge (DOD) is chosen. If both batteries are below their DODs, then the battery that has the lowest capacity is chosen. Otherwise, if both batteries are charged above their DOD, then no battery becomes the Target. The possibility of no battery being the Target at first seems inappropriate. However, the batteries will have the longest life if they are only charged after sufficiently being discharged, i.e. reaching an appropriate depth of discharge.

Figure 25. Determine Target Battery.

5.   Charging Methods

Figure 26 shows the charging overview questions which occur prior to attempting to charge a battery. First, the satellite must not be in eclipse in order to allow charging. Also, there must be a Target. Finally, if charging is appropriate, one of two methods are used, depending on the charge state of the Target battery. The two methods are overcharge and recharge.

Charge Battery

```
                              In Eclipse?
                         Yes              No

                                                  Target
                                          Yes      =
                                                  (A OR B)?
                                                      No

                                     Target
                                     need Overcharge
                                     OR
                               Yes   Count(Target)
                                     >
                                     MaxCount?
                                         No

  Turn OFF charge      Charge Target     Charge Target      Perform NO
  switches to both     Using Overcharge  Using Recharge     battery
  batteries            Method.           Method.            charging.
  (Maintain all charge
  state variables).

                          End.
```
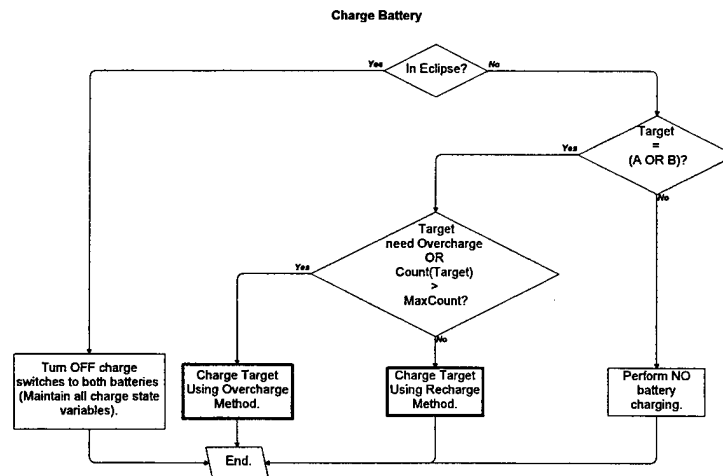
Figure 26. Battery Charge.

a. Overcharge

A Target battery is charged using the overcharge method when its charge state is unknown or it has already been recharged a maximum allowable number of times. The overcharge method, shown in Figure 27, keeps the Target charged until a voltage threshold has been reached. This voltage threshold depends on the temperature of the battery. Once this threshold has been reached, a timer is set. Typical overcharging of a battery is based on a time which depends on the allowable power mode of the spacecraft (Low, Standby, and Normal). By charging a set amount of time, a battery can be guaranteed to reach beyond its 100% capacity (regarding the amount of charge placed into the battery); and thus, at the completion of this overcharge the battery can be considered 100% full, containing 100% of its capacity.

75
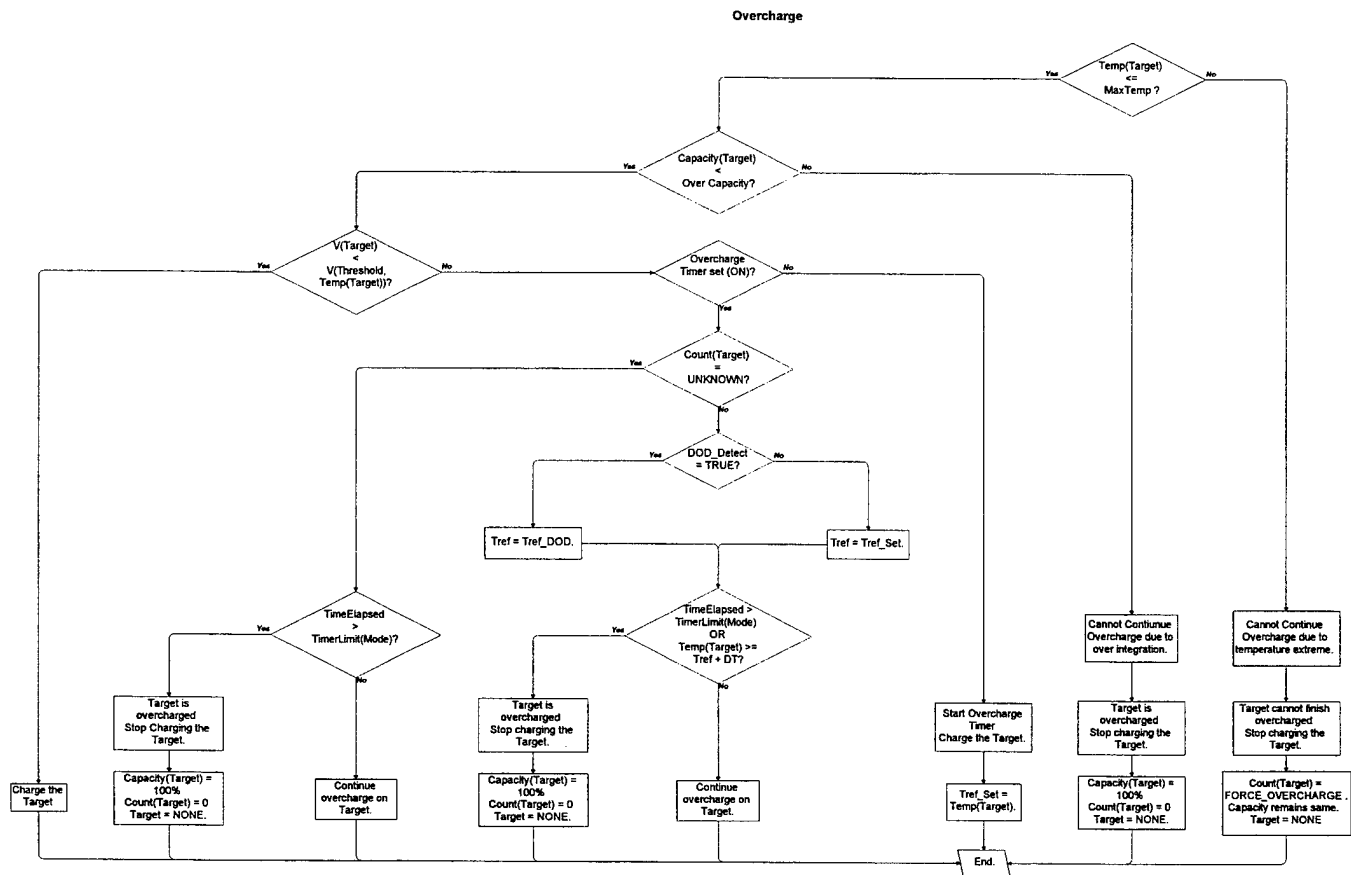
Figure 27. Battery Overcharge.

b. Recharge

A Target battery is charged using the recharge method when its charge state is known and it has not been recharged a maximum allowable number of times. The recharge method, shown in Figure 28, keeps the Target charged until its full capacity has been reached. This is performed using current integration in order to monitor the amount of current that enters the battery.
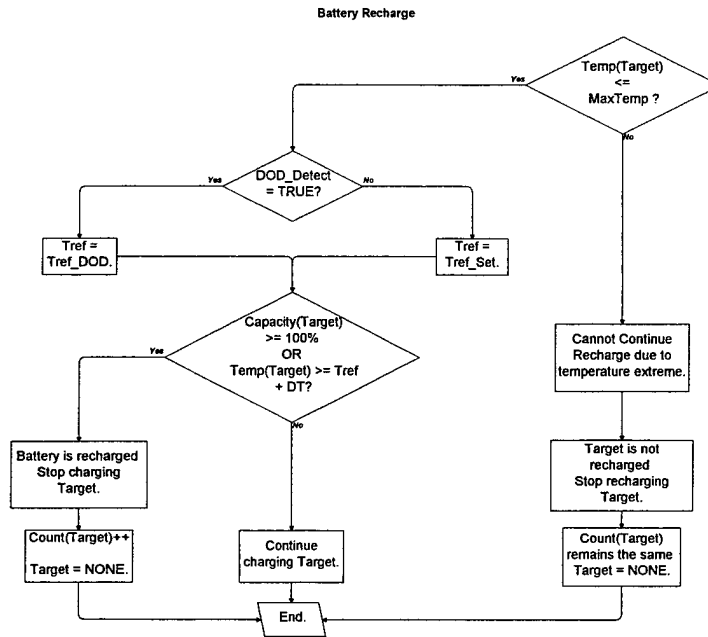
Figure 28. Battery Recharge.

6. Battery Mode

The operational mode of PANSAT depends on the charge state of the batteries which is maintained by the BCM. The operational mode indicates how much known stored energy is in the batteries at any time, and thus dictates how much power should be consumed by spacecraft operations. Figure 29 shows the decisions made to determine the mode.
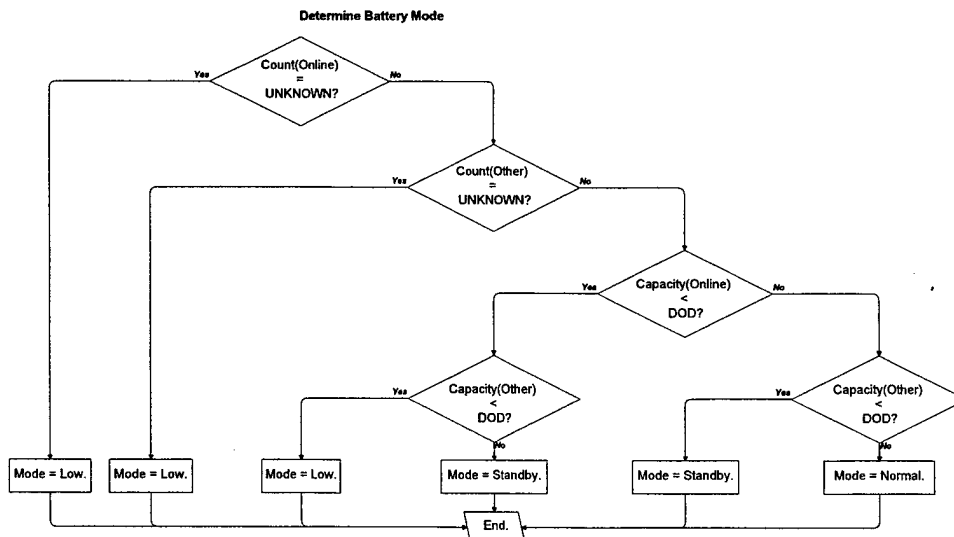
Figure 29. Battery Mode.

77

D. GROUND STATION COMMAND INTERFACE

After deployment, PANSAT only accepts requests received by RF transmissions. The ROM Boot Loader understands a very limited set of commands. The goal of the Boot Loader is to load the secondary loader while maintaining the spacecraft subsystems (in particular the batteries) in the most simple manner possible. Commands are sent from the NPS ground station as packets of data which are requests to the satellite to perform some action. The satellite responds to all commands with an acknowledgement packet which may contain extra information depending on the command sent. Table 22 shows the commands accepted during this initial stage of operation. The command encoding was designed so that each command has 4 bit differences between any other command. Even though the packets are CRC encoded in order to determine if there are bit errors, it is easy to also apply an encoding mechanism onto the commands to provide further error prevention.

Command	Encoding	Description
Confirm	0x55	Confirm a control command.
Control	0x5A	Request satellite to perform a particular operation.
Execute	0xA5	Transfer control to uploaded software (secondary loader).
Get Parameters	0xAA	Get parameters of A/D, BCM, hardware scenarios, RAM wash, time.
Load	0x66	Load a block of data into a specific area of RAM.
Map	0x69	Send Address/Data Map for future Load commands.
Reset	0x96	Stop the System Controller, forcing alternate to Reset.
Set Parameters	0x99	Set parameters of A/D, BCM, hardware scenarios, RAM wash, time.
Status	0x00	Send state-of-health (SOH) information.
Status Log Clear	0x0F	
Status Log Read	0xF0	Send recorded SOH.
Verify	0xFF	Verify a block of memory in RAM.
Unknown	?	Command received but is corrupt.

Table 22. ROM Boot Loader Commands.

1. Command Packet Protocol

The data field within a command packet is limited to 262 bytes of data, regardless of the command. This odd number was chosen to allow a paragraph aligned, 256 byte block of data to be sent with the Load command to upload code and data images. The bytes in the data field are numbered 0 through 261. The data field format depends on the command.

Byte[0]	Byte[1] - Byte[261]
Command	Data (relating to the command)

Table 23. Command Packet Data Field.

2. Commands

a. Confirm

The Confirm command is in response to the spacecraft receiving a command that requires confirmation, in which the spacecraft then sends an acknowledgment that requires *confirmation*.

b. Control

The Control command contains another command within the request packet that specifies some action the satellite should perform and return any data (or at least an acknowledge of having completed the action) back to the ground station. Permissible commands are shown in Table 20.

c. Execute

The Execute command also contains an absolute memory address which is an instruction pointer which contains the address of uploaded software in which transfer of control will be given. The spacecraft will send an acknowledgement before it *executes*, requiring the *confirm* command to be sent from the ground station.

d. Get Parameters

This command requests the spacecraft to send down all the parameters that describe and regulate the activities of the autonomous control algorithms, e.g. RF, Modem configuration, A/D gather and record rates, RAM wash rate, and Battery Control Monitor configuration.

e. Load

The Load command contains an address followed by a block of data. The address is the beginning address where the data should be stored. Thus, the first data byte will be stored at the address, the next data byte will be stored at the next larger address, etc. This command allows an arbitrary memory image to be transferred from the NPS ground station to PANSAT. Normally, this command is used to upload an image of the secondary loader which will take over the work of the Boot Loader.

The Load command uploads data pages which are up to 256 bytes in size. The absolute address at which the data is to be loaded is given. Upon successfully receiving a page without errors and storing the page into RAM without errors, the page address is recorded in a list. This list of successful loads is used later when verify commands are given.

f. Map

The Map command is used prior to a sequence of Load commands in order to let the spacecraft know about the number, location, and size of data blocks. All Map commands are acknowledged by PANSAT immediately following successful reception. If a negative acknowledgement is

sent (or if no acknowledgement is sent), the Map command needs repeating. Multiple Map commands may be necessary if a large number of Loads follow.

g. Reset

The Reset command forces the System Controller to enter a halt state. This will cause the SC to fail to update the watchdog timer and thereby causing it to be powered down by the EPS. This command causes the spacecraft to first send an acknowledgement of receiving the command; the *confirm* command must then be sent to perform the reset.

h. Set Parameters

This command indicates to the spacecraft that there are new parameters to set within the spacecraft which affect the activities of the autonomous control algorithms, e.g. RF, Modem configuration, A/D gather and record rates, RAM wash rate, and Battery Control Monitor configuration. This command causes the spacecraft to first send an acknowledgement of receiving the command; the *confirm* command must then be sent to perform the parameter setting.

i. Status

The Status command requests the satellite to send a complete state-of-health packet back to NPS. This information contains sensor data acquired by the A/D acquisition system as well as various software variables and parameters which describe the operational mode of PANSAT.

j. Status Log Clear

This command requests the spacecraft to clear all of the recorded status records on the mass storage. This command causes the spacecraft to first send an acknowledgement of receiving the command; the *confirm* command must then be sent to perform the clearing of records.

k. Status Log Read

This command tells the spacecraft to send down all of the recorded status records that have been saved to the mass storage.

l. Verify

The Verify command is accompanied by an absolute memory address, which specifies a block of data as used in the load command. Upon receipt of this command, PANSAT will check its list of receive data blocks from earlier Map and Load commands. A response is returned, in the form of a data block starting address, indicating which data blocks failed and need repeat transmission.

m. Unknown

In the event that the spacecraft actually receives an incoming packet in the form of a command but is unable to make exact sense of the command, this response is sent back to the NPS ground station, explaining that no action occurred.

3.   Loading Sequence

The loading of blocks of data from the ground station to the spacecraft is a complex sequence of operations, which is best viewed from a diagram. The sequence, Figure 30, is from the point of view of the ground station, sending blocks of data to the spacecraft. To begin, maps are built that describe all of the data blocks to be sent up; then the maps are sent using the Map command. Each Map command is acknowledged by the spacecraft before the next one is sent. Next, the blocks are prepared and, using the Load command, each block is sent up without any acknowledgement from the spacecraft. Finally, verifications are prepared. For each Verify command sent, the spacecraft is to send down an immediate response. The response may be in the form of multiple packets (if there were many Load errors). Following this response from the spacecraft, for each Load error identified, the block is prepared and resent to the spacecraft. The verification process is complete when the spacecraft returns a response indicating that there are no known Load errors.
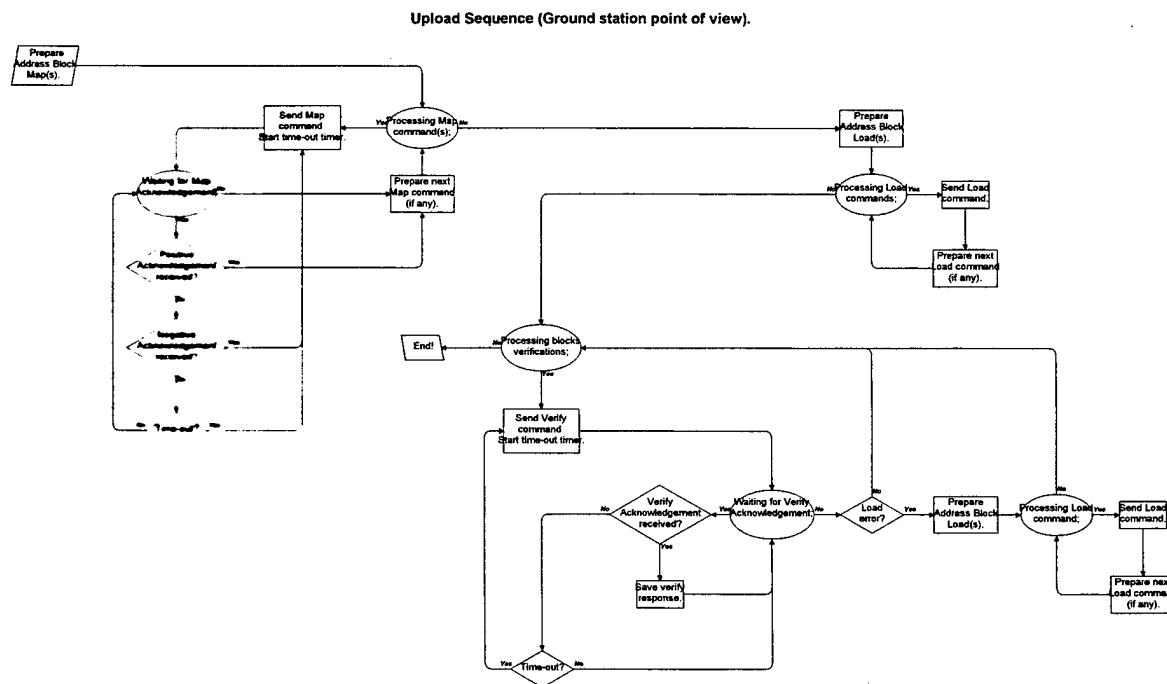
Figure 30. Block Loading Sequence.

81

E. SCENARIO CHECKS

On a regular basis during the execution of the Boot Loader, hardware and software sensors are used to determine whether or not various hardware subsystems of PANSAT are functioning correctly. Many of these checks are directly related to the communication systems. Since a lack of communication from NPS most likely indicates that hardware on board the satellite is not functioning correctly or is not in the mode presumed, symptoms that show this type of behavior are monitored. In the event that such a symptom is found, the spacecraft either programs the hardware to another configuration or uses an entirely different piece of hardware. Figure 31 shows a flow diagram indicating the various symptoms which are monitored and the type of cure used.

Figure 31. Scenario Check.
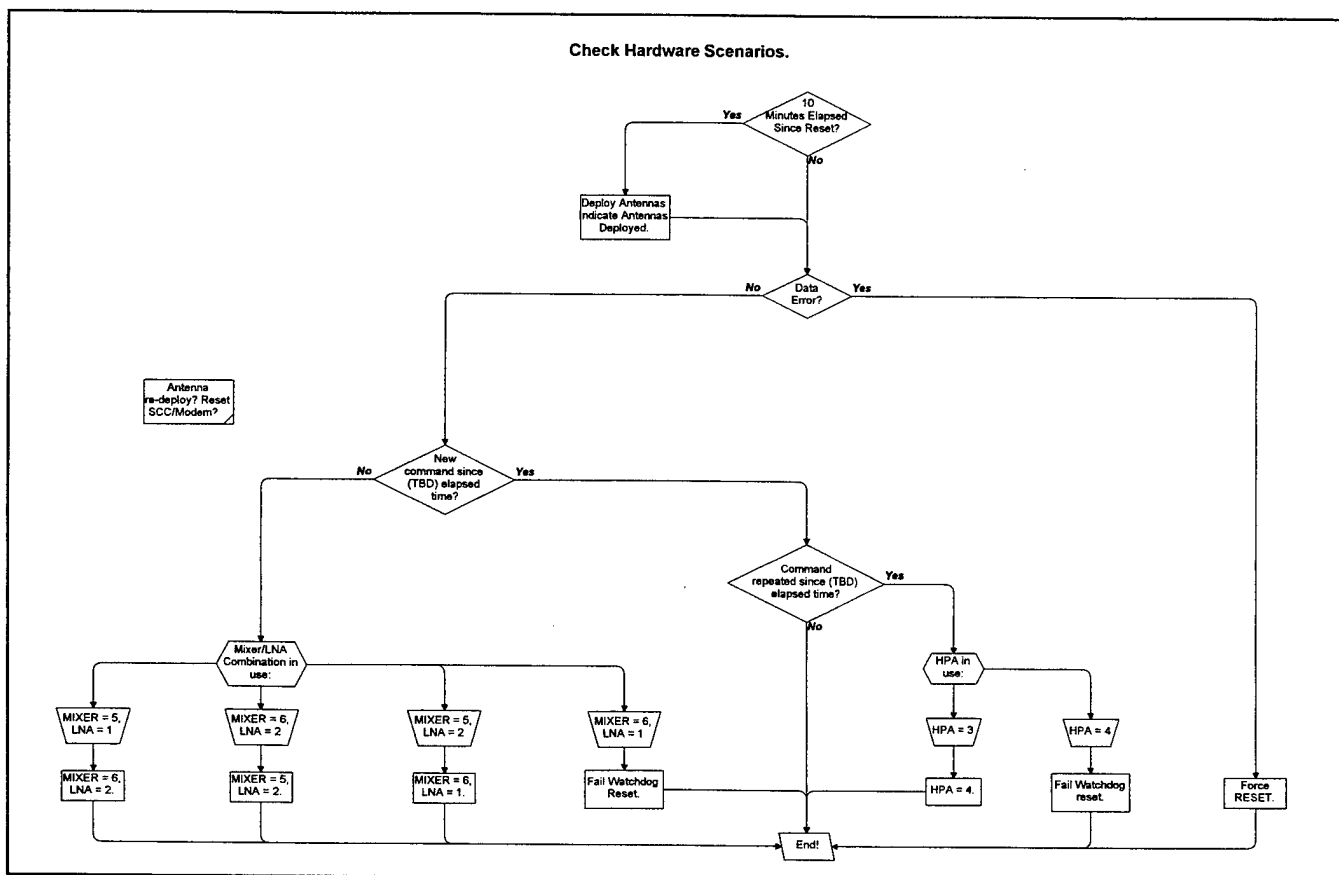
VII. RESULTS, RECOMMENDATIONS, AND CONCLUSION

A. RESULTS

1. Printed Circuit Board

The System Controller described in this thesis was fabricated on a printed circuit board. The layout of the board was performed by David Rigmaiden of the Space Systems Academic Group with the assistance of the author and the schematics generated with this document. Accel Technology's P-CAD/Tango software was used as the layout tool. The result is a six layer board suitable for space flight. Details of the layout and board manufacturing are beyond the scope of this document.

The printed circuit board was component stuffed by Rigmaiden and was tested by the author. The testing involved the verification of each circuit element on the PCB. Appropriate voltage levels were measured and were in accordance with expected values. Higher-level testing was accomplished by means of software control and reading back of data if possible. Various voltage meters, oscilliscopes, and an in-circuit emulator (ICE) connected to a general purpose computer were used to determine that the system was operating properly.

2. Use of In-Circuit Emulator

The Microtek MICE-III ICE was used to emulate the M80C186XL during initial tests. The microprocessor was removed from the printed circuit board, and the in-circuit emulator connected to the CPU's socket. This setup is shown in Figure 32. The emulator provided a means to run a program in emulation RAM and emulation ROM; later, the actual RAM and ROM were tested by placing these components onto the actual printed circuit board. The emulator allows rapid modification of the test programs without the requirement to re-program the system EPROM. The emulator also provides an interface that allows single stepping through test programs at the machine level or at a higher level (e.g. C program statements). Furthermore, sophisticated breakpoints based on hardware or software conditions are programmable.

83
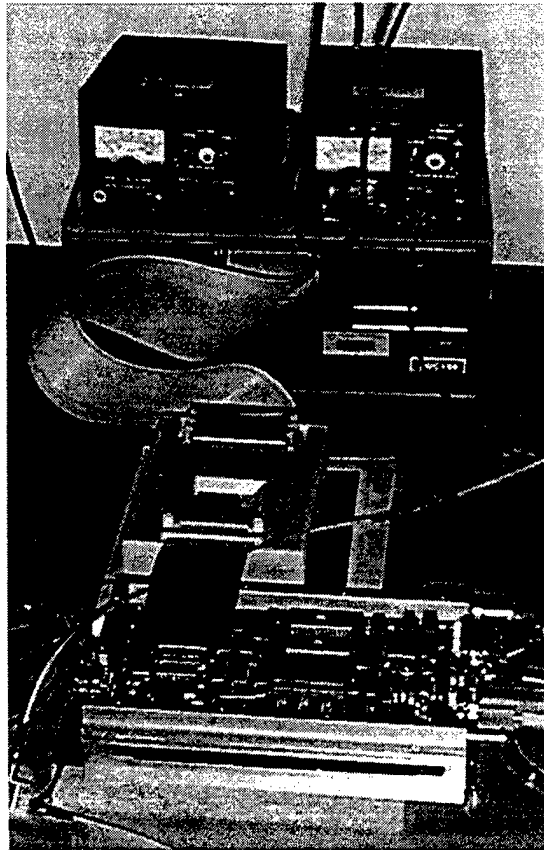
Figure 32. In-Circuit Emulator.

The emulator exhibited unusual behavior during testing with programmed ROMs. The emulator was unable to drive the memory system with correct voltage levels; and thus, the ROMs on the printed circuit board were not correctly stimulated and did not operate. This was of little consequence to development however, since ROMs are needed only after significant testing and development has occurred. This unusual behavior is attributed to the various families of interface logic that differ between the ICE and the board.

3.   Software

Device drivers were used to test the hardware peripherals of the System Controller as well as the electronics of the satellite subsystems. Device drivers were tested first individually while observing that the hardware was driven as desired. Next, device drivers were tested while operating together as multiple interrrupt service routines while noting that the hardware was driven as desired. Higher-level software routines were tested using the same methods as described for the device drivers. However, higher-level software tests ran autonomously, logging data to a general purpose computer which could be examined during and after the tests to ensure correctness of operation.

84

4. Testing

Tests were performed on the fabricated System Controller to evaluate the operation of the hardware as well as verify the software that runs on the hardware. The results were all positive. Appendix K lists the tests performed during different stages of the System Controller construction.

B. RECOMMENDATIONS

The design of the System Controller glue-logic can be replaced with a programmable logic device, such as a programmable gate array (PGA). This would reduce PCB area and power consumption. The Space Systems Academic Group needs to investigate the acquisition of tools to simulate and test as well as program such devices.

An additional design change would be to not only incorporate the glue logic, but also the EDAC controller and memory buffers into the same PGA. This would further reduce power consumption and PCB area. Such a controller could use additional M80C186 status bits (S2 - S0). These status bits, along with the Bus High Enable and A0 signals are available turing the first half of the first T state of the microprocessor. These signals can identify if word write operations are about to occur in which the "read and correct" can be eliminated and the associated error flags may also be eliminated. This would also reduce power consumption since the RAM would be accessed less.

The peripheral control bus (PCB) of the spacecraft requires too much supervision by the CPU in order to be effective for a data bus that requires higher data transfer rates. A simple controller could be designed that accesses a small FIFO memory which stores subsystem address, sub-addresses, and data (for a PCB write operation). The CPU could quickly add elements to the FIFO and let the controller independently handle all of the PPI transactions. Another incoming FIFO would be used to store data read from the PCB; it would be accessed by the CPU to gather incoming data.

C. CONCLUSION

The System Controller hardware and embedded software described in this thesis provide PANSAT with a reliable digital computer suitable for use in the LEO environment. The system is capable of autonomously controlling the spacecraft after launch and reset conditions. The system meets the design requirements by using a small number of readily available, reasonably priced components.

APPENDIX A. HARDWARE SCHEMATICS

This appendix contains the detailed schematics for the System Controller hardware. The following drawings are included:

Description	Drawing (Figure)
System Controller - CPU and Data/Address Buffers	Figure 33
System Controller - Memory	Figure 34
System Controller - A/D and SCC	Figure 35
System Controller - PCB, Power Detect, and Power Supply	Figure 36

Figure 33. System Controller Schematic 1.

88

Figure 34. System Controller Schematic 2.

89
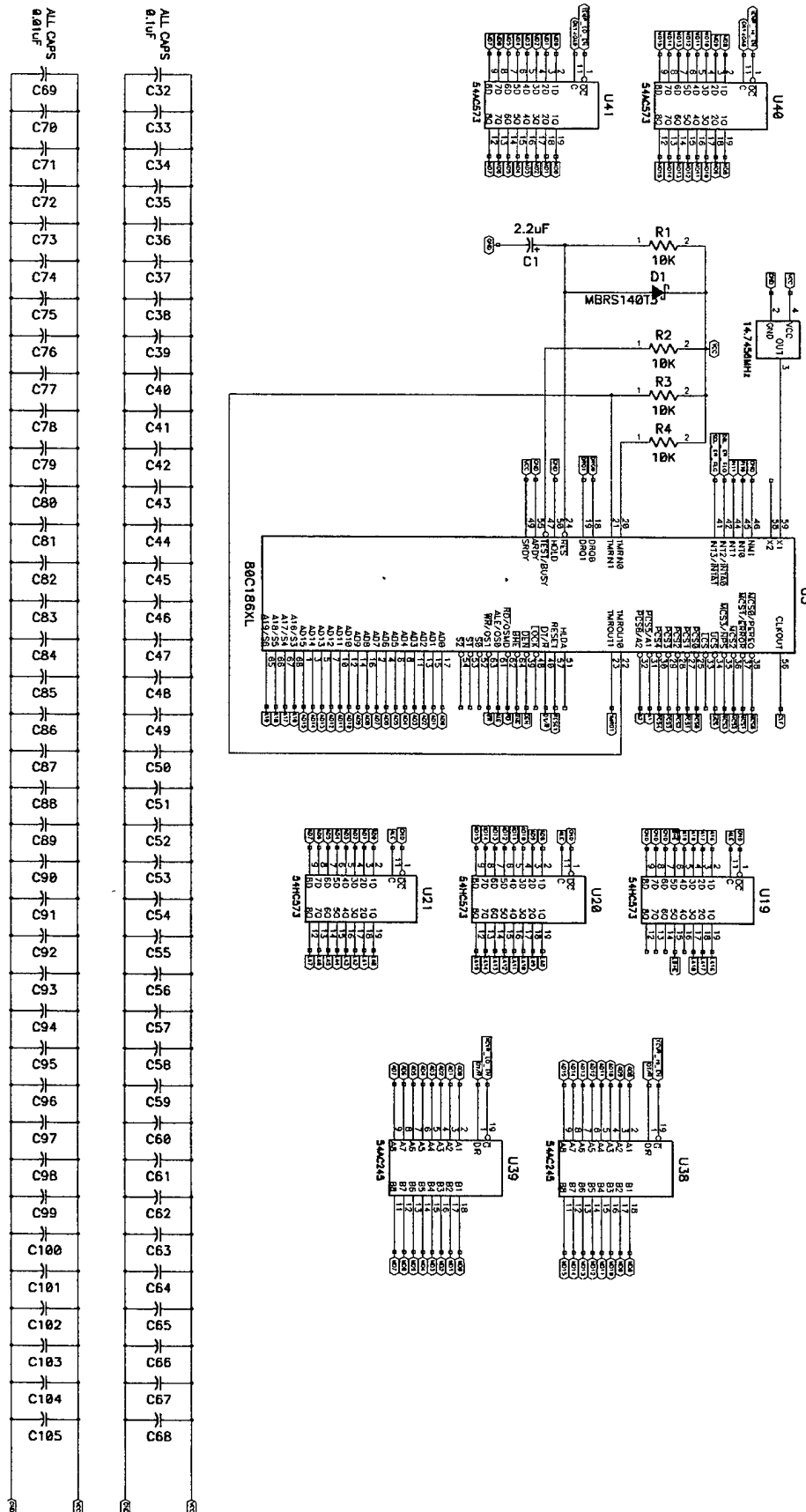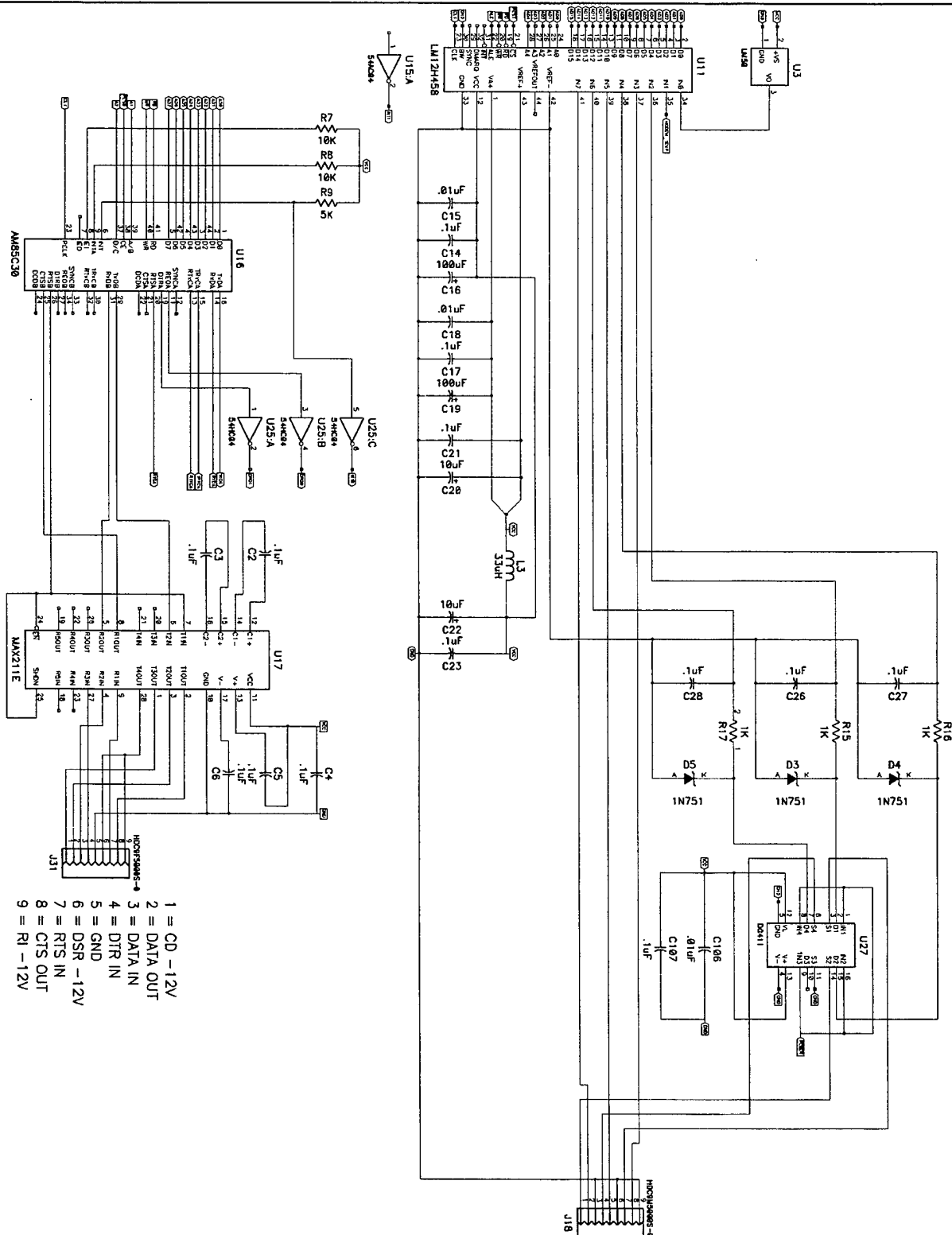
Figure 35. System Controller Schematic 3.

Figure 36. System Controller Schematic 4.

91

APPENDIX B. SYSTEM CONTROLLER CONNECTOR PIN-OUTS

This appendix contains the pin-outs for all of the connectors of a System Controller. The connectors are the 9-pin analog input signals, the 9-pin RS-232 serial test port interface, the 25-pin Peripheral Control Bus, and the 37-pin Modem interface.

Pin Number	Name	Description
1	D0	Data Bit 0
2	D1	Data Bit 1
3	D2	Data Bit 2
4	D3	Data Bit 3
5	D4	Data Bit 4
6	D5	Data Bit 5
7	D6	Data Bit 6
8	D7	Data Bit 7
9	S0	System Select Bit 0
10	S1	System Select Bit 1
11	S2	System Select Bit 2
12	GND	Ground
13	SC_A	System Controller Active
14	VCC	PCB +5 V Power
15	SW_+12	Switched +12 V Power
16	GND	Ground
17	*Unused*	*Unused*
18	RTSA	
19	PC1	
20	RF_EN	RF Enable
21	$\overline{RD}$	PCB Read
22	$\overline{WR}$	PCB Write
23	A1	Sub-address Bit 1
24	A0	Sub-address Bit 0
25	S3	System Select Bit 3

Table 24. Peripheral Control Bus Connector.

Pin Number	Name	Description
1	IOA6	Address Bit 6
2	IOA4	Address Bit 4
3	IOA2	Address Bit 2
4	$\overline{\text{PA100}}$	PA100 Chip Select
5	GND	Ground
6	GND	Ground
7	IOD7	Data Bit 7
8	IOD5	Data Bit 5
9	IOD3	Data Bit 3
10	IOD1	Data Bit 1
11	GND	Ground
12	VCC	Power To Buffers
13	GND	Ground
14	$\overline{\text{IORD}}$	Read
15	MTXDATA	Modem TX Data
16	MRXDATA	Modem RX Data
17	+5 V	Switched +5 V
18	+5 V	Switched +5 V
19	MODEM_TEMP	Modem Temperature
20	IOA5	Address Bit 5
21	IOA3	Address Bit 3
22	IOA1	Address Bit 1
23	$\overline{\text{LATCH}}$	Latch Chip Select
24	GND	Ground
25	GND	Ground
26	IOD6	Data Bit 6
27	IOD4	Data Bit 4
28	IOD2	Data Bit 2
29	IOD0	Data Bit 0
30	GND	Ground
31	VCC	Power To Buffers
32	GND	Ground
33	$\overline{\text{IOWR}}$	Write
34	MTXCLK	Modem TX Clock
35	MRXCLK	Modem RX Clock
36	+5 V	Switched +5 V
37	+5 V	Switched +5 V

Table 25. Modem Connector.

Pin Number	Function
1	TMUXA+
2	TMUXA- (ground)
3	Ground
4	TMUXB+
5	TMUXB- (ground)
6	Ground
7	EPS+
8	EPS- (ground)
9	Ground

Table 26. Analog Signals Connector.

Pin Number	Name	Function	Direction
1	CD	Carrier Detect	Out
2	Tx	Transmit Data	Out
3	Rx	Receive Data	In
4	DTR	Data Terminal Ready	In
5	GND	Ground	
6	DSR	Data Set Ready	Out
7	RTS	Ready To Send	In
8	CTS	Clear To Send	Out
9	RI	Ring Indicator	Out

Table 27. RS-232 Serial Port Connector.

APPENDIX C. CIRCUIT BOARD BILL OF MATERIALS

Type	Pattern	Value	Designators
CAP0805	CAP0805	.01uF	C10 C100 C101 C102 C103 C104 C105 C106 C15 C18 C69 C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81 C82 C83 C84 C85 C86 C87 C88 C89 C90 C91 C92 C93 C94 C95 C96 C97 C98 C99
		330pF	C11
CAP1206	CAP1206	.1uF	C107 C108 C109 C14 C17 C2 C21 C23 C26 C27 C28 C3 C32 C33 C34 C35 C36 C37 C38 C39 C4 C40 C41 C42 C43 C44 C45 C46 C47 C48 C49 C5 C50 C51 C52 C53 C54 C55 C56 C57 C58 C59 C6 C60 C61 C62 C63 C64 C65 C66 C67 C68 C7 C9
CTX32	CTX32	33uH	L3
DB25RM	DB25RM	HDC25M5000S-0	J2
DB37F	DB37F	HDC37F32000S-0	MJ
DB9RF	DB9RF	HDC9F5000S-0	J31

DB9RM	DB9RM	HDC9M5000S-0	J18
54HC04	DIP14	54AC04	U15
		54HC04	U25
54HC08	DIP14	54AC08	U1 U12 U33
		54HC08	U18 U35
54HC125	DIP14	54HC125	U2
54HC32	DIP14	54AC32	U34 U4
		54HC32	U32 U37
54HC74	DIP14	54AC74	U29
DG411	DIP16	DG411	U27
54HC245	DIP20	54AC245	U38 U39 U8
		54HC245	U10 U22 U23
54HC573	DIP20	54AC573	U40 U41
		54HC573	U19 U20 U21 U7
54HC574	DIP20	54AC574	U42
27C256-120	DIP28		U36 U43
1N751	DO-35	{Value}	D3 D4 D5
LM50	LM50	LM50	U3
80C186XL	P186XL		U5
ACS630MS	P630		U26
PCAP3528	PCAP3528	2.2uF	C1 C13
PCAP6032	PCAP6032	10uF	C110 C20 C22
PCAP7343	PCAP7343	100uF	C12 C16 C19 C8
CTX33-2	PCTXFORM	33uH	L2
CTX50-2	PCTXFORM	50uH	L1
AM85C30	PPLC44		U16
IS82C55A	PPLC44		U14
LM12H458	PPLC44		U11
MSM8256	PSRAM		U30 U31 U44

MAX744A	PWSO16		U13
MAX211E	PWSO28		U17
RES1206	RES1206	10K	R1 R2 R3 R4 R5 R6 R7 R8
		120	R22 R23 R24 R25
		147K	R10
		1K	R15 R16 R17
		1M	R12
		21.5K	R11
		511K	R14
		5K	R13 R21 R9
MBRS140T3	SMB	MBRS140T3	D1
MBRS340T3	SMC	MBRS340T3	D2
TPS2013	SO8	TPS2013	U6
ICL8212	TO-99	ICL8212	U24
XTAL-OSC	XTAL-OSC	14.7456MHz	U9

APPENDIX D. PERIPHERAL CONTROL BUS PROGRAMMABLE PERIPHERAL INTERFACE PORT CONFIGURATION

This appendix contains a detailed description of the programmable peripheral interface (PPI - 8255) that is used to control the peripheral control bus of the spacecraft.

Bit	Function
7	Read (active LOW)
6	Write (active LOW)
5	Sub-address 1
4	Sub-address 0
3	Select 3
2	Select 2
1	Select 1
0	Select 0

Table 28. Port B Assignments for PPI (Address and Read/Write Control).

Port Address	Port Name	Usage
0x100	Port A	Bi-directional using Port C for handshake.
0x102	Port B	Address selection and Read & Write strobes.
0x104	Port C	Handshaking for Port A, EDAC and Modem Control.
0x106	Control	Control register for this PPI.

Table 29. PPI Port Usage.

Bit	Function
7	Handshake (unused)
6	Handshake
5	Handshake (unused)
4	Handshake
3	Handshake (unused)
2	EDAC Error Acknowledge (active LOW)
1	Read Latch Enable (active LOW)
0	Modem Power (active LOW)

Table 30. Port C Assignments for PPI (Handshaking and Control).

APPENDIX E. ELECTRICAL POWER SYSTEM PORT CONFIGURATION

This appendix contains a detailed description of the Electrical Power System (EPS) ports.

Port	Address	Subaddress	Bit	Contents
Port 0	8	0	7	Battery A Charge (*1 =enable*)
			6	Battery A Discharge (*1 = enable*)
			5	Battery A Online (*1 = enable*)
			4	Battery A Trickle Charge (*1 = enable*)
			3	Mass Storage A Power (*1 = enable*)
			2	Temperature MUX A Power (*1 = enable*)
			1	*unused*
			0	Battery A Heater (*1 = enable*)

Table 31. EPS Port 0.

Port	Address	Subaddress	Bit	Contents
Port 1	8	1	7	Low Cell Voltage MUX Select 2
			6	Low Cell Voltage MUX Select 1
			5	Low Cell Voltage MUX Select 0
			4	Low Cell Voltage Enable (*1 = enable*)
			3	Medium Cell MUX Select 2
			2	Medium Cell MUX Select 1
			1	Medium Cell MUX Select 0
			0	Medium Cell Voltage Enable (*1 = enable*)

Table 32. EPS Control Port 1.

Port	Address	Subaddress	Bit	Contents
Port 1	8	2	7	*unused*
			6	Battery B Heater (*1 = enable*)
			5	RF Power (*1 = enable*)
			4	Temperature MUX B Power (*1 = enable*)
			3	Mass Storage B Power (*1 = enable*)
			2	Antenna Release (*1 = enable*)
			1	Solar Panel Current Inhibit (*0 = inhibit, 1 = enable*)
			0	Solar Panel Current Strobe

Table 33. EPS Control Port 2.

103

Port	Address	Subaddress	Bit	Contents
Port 3	8	3	7	Current Select, or High Cell Voltage Select 3
			6	Current Select, or High Cell Voltage Select 2
			5	Current Select, or High Cell Voltage Select 1
			4	Current Select, or High Cell Voltage Select 0
			3	High Cell Voltage MUX Select 2
			2	High Cell Voltage MUX Select 1
			1	High Cell Voltage MUX Select 0
			0	High Cell Voltage Enable (*1 = enable*)

Table 34. EPS Control Port 2.

Port	Address	Subaddress	Bit	Contents
Port 5	9	1	7	*unused*
			6	*unused*
			5	*unused*
			4	*unused*
			3	*unused*
			2	*unused*
			1	Battery A Current Sense (0 = discharging)
			0	Battery B Current Sense (0 = discharging)

Table 35. Read-back Port 5.

Port	Address	Subaddress	Bit	Contents
Port 6	9	2	7	Battery B Charge (*1 =enable*)
			6	Battery B Discharge (*1 = enable*)
			5	Battery B Online (*1 = enable*)
			4	Battery B Trickle Charge (*1 = enable*)
			3	*unused*
			2	*unused*
			1	*unused*
			0	*unused*

Table 36. EPS Control Port 6.

Battery Cell	MUX Select Control (Port 3)	
0A	0000 1110	[0x0E]
1A	0000 1001	[0x09]
0B	0000 1101	[0x0D]
1B	0000 1011	[0x0B]

Table 37. EPS Low Cell Voltage Selections.

Battery Cell	MUX Select Control (Port 3)	
2A	1000 1111	[0x8F]
3A	1100 1111	[0xCF]
4A	1010 1111	[0xAF]
2B	1110 1111	[0xEF]
3B	1001 1111	[0x9F]
4B	1101 1111	[0xDF]

Table 38. EPS Medium Cell Voltage Selections.

Battery Cell	MUX Select Control (Port 2)	
5A	1010 0000	[0xA0]
6A	1110 0000	[0xE0]
7A	1001 0000	[0x90]
5B	1101 0000	[0xD0]
6B	1011 0000	[0xB0]
7B	1111 0000	[0xF0]
8A	1100 0011	[0xC3]
9A	1100 1011	[0xCB]
8B	1000 1111	[0x8F]
9B	1000 0111	[0x87]

Table 39. EPS High Cell Voltage Selections

Current Selection	Label #	MUX Select Control (Port 2)		Port 3	
Solar Panel:		0000 0000	[00]	0000 1100	[0x0C]
Solar Panel:		0000 1000	[08]	0000 1100	[0x0C]
Solar Panel:		0000 0100	[04]	0000 1100	[0x0C]
Solar Panel:		0000 1100	[0x0C]	0000 1100	[0x0C]
Solar Panel:		0000 0010	[02]	0000 1100	[0x0C]
Solar Panel:		0000 1010	[0x0A]	0000 1100	[0x0C]
Solar Panel:		0000 0110	[06]	0000 1100	[0x0C]
Solar Panel:		0000 1110	[0x0E]	0000 1100	[0x0C]
Solar Panel Bus		0000 0101	[05]	0000 1010	[0x0A]
Battery A		0000 1001	[09]	0000 1010	[0x0A]
Battery B		0000 0001	[01]	0000 1010	[0x0A]

Table 40. EPS Current Selections.

105

APPENDIX F. A/D ACQUISITION

Period	Cycle	Set	A/D IN0 (DCS)	A/D IN1 (Modem)	A/D IN2 (EPS)	A/D IN4 (TMUXA)	A/D IN6 (TMUXB)
0	0	0	DCS	Modem	SC Current	0	0
1	1				Battery A Cell 0 Voltage	1	1
2	2				Battery A Cell 1 Voltage	2	2
3	0	1			Battery A Current	3	3
4	1				Battery A Cell 2 Voltage	4	4
5	2				Battery A Cell 3 Voltage	5	5
6	0	2			Battery B Current	6	6
7	1				Battery A Cell 4 Voltage	7	7
8	2				Battery A Cell 5 Voltage	8	8
9	0	3			SC Current .	9	9
10	1				Battery A Cell 6 Voltage	10	10
11	2				Battery A Cell 7 Voltage	11	11
12	0	4			Battery A Current	12	12
13	1				Battery A Cell 8 Voltage	13	13
14	2				Battery B Cell 0 Voltage	14	14
15	0	5			Battery B Current	15	15
16	1				Battery B Cell 1 Voltage	16	16
17	2				Battery B Cell 2 Voltage	17	17
18	0	6			SC Current	18	18
19	1				Battery B Cell 3 Voltage	19	19
20	2				Battery B Cell 4 Voltage	20	20
21	0	7			Battery A Current	21	21
22	1				Battery B Cell 5 Voltage	22	22
23	2				Battery B Cell 6 Voltage	23	23
24	0	8			Battery B Current	24	24
25	1				Battery B Cell 7 Voltage	25	25
26	2				Battery B Cell 8 Voltage	26	26
27	0	9			SC Current	27	27
28	1				Spacecraft Bus Voltage	28	28
29	2				Solar Panel 0 Current	29	29
30	0	10			Battery A Current	30	30
31	1				Solar Panel 1 Current	31	31
32	2				Solar Panel 2 Current		
33	0	11			Battery B Current		
34	1				Solar Panel 3 Current		
35	2				Solar Panel 4 Current		
36	0	12			SC Current		
37	1				Solar Panel 5 Current		
38	2				Solar Panel 6 Current		
39	0	13			Battery A Current		
40	1				Battery B Current		
41	2				Solar Panel 7 Current		

Table 41. A/D Conversion Schedule.

APPENDIX G. THERMISTOR TEMPERATURE CONVERSIONS

To simplify temperature conversions for the Omega 440048 thermistors, a table is used to perform a binary search where a maximum of seven compares are needed for a lookup. The table is used by taking the value from the A/D converter and finding the closest match in the table. The position of the match in the table indicates the temperature of the sensor. The following discussion describes how the conversion process works.

$$T = \left[\frac{1}{A + B * \ln(R) + C * \left[\ln(R)\right]^3} \right] - 273.15 \tag{10}$$

The equation shown above uses three coefficients which were calculated using the temperature range suggestions from the Omege Temperature Sensor manual [Ref. X]. A short MATLAB program (given below) was created to determine these coefficients based on the assumption that the most accurate temperature conversions are needed in the temperature ranges between 0^oC to 30^oC. The ability to change the coefficients means that the conversion formula can be tailored to have the best conversion for a certain temperature range. In doing so, three temperatures must be selected when using the program. One temperature must be below the range, another within the range, and the third above the range. Furthermore, there can be no more than 100^oC between the two extremes, and each successive temperature can be no more than 60^oC apart. The temperatures used are -30^oC, 20^oC, and 60^oC. The corresponding resistances based on the generic lookup conversion provided by Omega are 481 kΩ, 37.3 kΩ, and 7.599 kΩ . The values for the coefficients are A=9.306x10^{-4}, B= 2.218x10^{-4}, and C=1.253x10^{-7}.

```
T1 = -30 + 273.15;
T2 = 20 + 273.15;
T3 = 60 + 273.15;
T = [1/T1; 1/T2; 1/T3];

R1 = 481.0E3;
R2 = 37.3E3;
R3 = 7599;
R = [1 1 1; log(R1) log(R2) log(R3); (log(R1))^3 (log(R2))^3
        (log(R3))^3]';

S = inv(R)*T;
S
```

One channel of each temperature multiplexing unit is reserved as a fixed 1% precision resistor is used as a calibration resistor to create a calibration current. The calibration current, I_C, is converted as follows.

$$I_C = \frac{V_C}{R_C} = \frac{N * \left(\frac{5}{4095}\right)}{R_C} = 0.001221 * \left(\frac{N}{R_C}\right)$$

where R_C is the resistance of a (fixed 1%)
Calibration Resistor equal to xxx Ω.

(11)

Thus, a particular thermistor resistance, R, is converted as follows.

$$R = \frac{V_R}{I_C} = \frac{N * \left(\frac{5}{4095}\right)}{I_C} = 0.001221 * \left(\frac{N}{I_C}\right)$$

(12)

For quick temperature conversions, a table lookup is used where the value N is an index into the table. This table is shown in Table 42. Assuming a constant calibration current, the table can remain static. If the calibration current changes significantly, the table can be recalculated simply by multiplying all of the entries by the change in the calibration current.

$$N = \frac{R * I_C}{\left(\frac{5}{4095}\right)} = 819 * R * I_C$$

(13)

Because of the resolution of the A/D converter, temperatures above 88° C do not have unique conversions from the A/D. Since 5 mV of noise are allowed within the system, temperatures above 44° C are not accurate to within one degree Celsius. Finally, a saturated reading from the A/D, i.e. N = 4095, corresponds to any low temperature below -30° C. Note, 4191 is above the limit of the A/D, yet corresponds to the next integral temperature below -30° C.

T (C)	A/D (N)		T (C)	A/D (N)		T (C)	A/D (N)
-30.0	3949		10.0	482		50.0	90
-29.0	3723		11.0	461		51.0	87
-28.0	3512		12.0	440		52.0	84
-27.0	3313		13.0	420		53.0	81
-26.0	3127		14.0	401		54.0	78
-25.0	2952		15.0	383		55.0	75
-24.0	2788		16.0	366		56.0	72
-23.0	2634		17.0	350		57.0	70
-22.0	2490		18.0	335		58.0	67
-21.0	2354		19.0	320		59.0	65
-20.0	2226		20.0	306		60.0	62
-19.0	2106		21.0	293		61.0	60
-18.0	1993		22.0	281		62.0	58
-17.0	1887		23.0	269		63.0	56
-16.0	1787		24.0	257		64.0	54
-15.0	1693		25.0	246		65.0	52
-14.0	1604		26.0	236		66.0	51
-13.0	1520		27.0	226		67.0	49
-12.0	1442		28.0	217		68.0	47
-11.0	1368		29.0	208		69.0	46
-10.0	1298		30.0	199		70.0	44
-9.0	1231		31.0	191		71.0	43
-8.0	1169		32.0	183		72.0	41
-7.0	1110		33.0	176		73.0	40
-6.0	1055		34.0	169		74.0	38
-5.0	1002		35.0	162		75.0	37
-4.0	953		36.0	156		76.0	36
-3.0	906		37.0	150		77.0	35
-2.0	862		38.0	144		78.0	34
-1.0	820		39.0	138		79.0	33
0.0	780		40.0	133		80.0	32
1.0	743		41.0	127		81.0	31
2.0	707		42.0	123		82.0	30
3.0	673		43.0	118		83.0	29
4.0	642		44.0	113		84.0	28
5.0	611		45.0	109		85.0	27
6.0	583		46.0	105		86.0	26
7.0	556		47.0	101		87.0	25
8.0	530		48.0	97		88.0	24
9.0	506		49.0	94			

Table 42. Thermistor Lookup Conversion Table.

111

APPENDIX H. SPACECRAFT COMMAND ENCODING

This appendix gives the command encoding details.

Byte[0]	Byte[1] - Byte[4]
0x05	Address

Table 43. Execute Command.

Byte[0]	Byte[1] - Byte[4]	Byte[5]	Byte[6] - Byte[261]
0x0A	Address	Count (0 ⇒ 256)	Code/data

Table 44. Load Command.

Byte[0]	Byte[1] - Byte[4]	Byte [5]	Byte[6] - Byte[9]	Byte [10]	..	Byte[256] - Byte[259]	Byte [260]
0x5A	Address	Size (0⇒256)	Address	Size (0⇒256)	..	Address	Size (0⇒256)

Table 45. Map Command.

Byte[0]	Byte[1] - Byte[3]
0xFF	0xAA, 0x55, 0xFF

Table 46. Reset Command.

Byte[0]
0x00

Table 47. Status Command.

Byte[0]
0xAA

Table 48. Verify Load Command.

APPENDIX I. SOFTWARE GENERATION FACILITIES

This appendix contains the software generation facilities for the ROM Boot Loader embedded software for the System Controller.

A PC compatible workstation using standard software generation tools creates the PANSAT Boot ROM software. The Microsoft C Compiler version 5.0 and the Microsoft Macro Assembler version 5.10 translate the software source code into object modules. The Make facility included with these code generators provides an automated method for generating the ROM image. Systems And Software, Xlink86 version 6.10e links all of the object modules into a relocatable code image. Systems And Software Xloc86 version 6.10 translates the relocatable image into absolute address code and data. Finally, Systems And Software PROM86 version 6.0a prepares a binary image suitable for the ROM.   The makefile, named *dcs.mak* and shown below, is responsible for identifying all the source modules and their dependencies which are required to build the entire ROM image for the embedded software.

```
####################################################################
#
#  DCS.MAK
#
#
#  Date           Who     What
#  ---------------+-------+--------------------------------------
#  25 March 1996  Jah     Creation
#
####################################################################


####################################################################
#
#  Compiler and Assembler options
#
####################################################################

#MAKEDIR = .

#
# Compiler options:   /c          no linking
#                     /AS         small model (64k code, 64k data)
#                     /Zp         pack structures on n boundary
#                     /Gs         no stack checking
#                     /Od         no optimizations
#                     /Oi         enable intrinsic functions
#                     /FPa    FP calls with altmath library
#                     /G1         use 80186 instructions
```

```
#                          /Zi          add symbolic debugging information

CFLAGS = /c /AS /Zp1 /Gs /Od /Oi /FPa /G1 /Zi

#
# Assembler options:/Mx      case-sensitive identifiers
#                          /Zi          add symbolic debugging information

AFLAGS = /Mx /Zi

#
# General (common) dependencies
#

GEN_DEPS = gen_defs.h gen_apis.h

OBJS =  dcs.obj ad.obj bcm.obj clock.obj edac.obj eps.obj gen_apis.obj modem.obj msu.obj\
                pcb.obj print.obj scc.obj stpi.obj terms.obj tlm.obj startup.obj

###############################################################################
#
#   Compilations
#
###############################################################################

dcs.omf:      dcs.abs
                         cv2omf dcs.abs to dcs.cv
                         prom86 dcs.cv to dcs.omf omf initdata
                    prom86 dcs.abs to dcs.bin ad(0F0000h, 0FFFFFh) initdata one

# Absolute address relocated image
dcs.abs:      dcs.lnk
                         xloc86 @dcs.loc

# Linked image
dcs.lnk:      $(OBJS)
                         xlink86 @dcs.lk

# source code modules
dcs.obj:           dcs.c dcs.h pcb.h $(GEN_DEPS)

ad.obj:            ad.c ad.h pcb.h tlm.h $(GEN_DEPS)

bcm.obj:            bcm.c bcm.h ad.h clock.h pcb.h tlm.h $(GEN_DEPS)

clock.obj:          clock.c edac.h $(GEN_DEPS)

edac.obj:           edac.c edac.h $(GEN_DEPS)

eps.obj:            eps.c eps.h pcb.h $(GEN_DEPS)

gen_apis.obj:   gen_apis.c gen_apis.h $(GEN_DEPS)

#int.obj:           int.c int.h $(GEN_DEPS)
```

```
modem.obj:        modem.c modem.h $(GEN_DEPS)

msu.obj:          msu.c msu.h pcb.h $(GEN_DEPS)

pcb.obj:          pcb.c pcb.h $(GEN_DEPS)

print.obj:        print.c print.h $(GEN_DEPS)

#rf.obj:           rf.c rf.h pcb.h $(GEN_DEPS)

scc.obj:          scc.c scc.h $(GEN_DEPS)

#scenario.obj:  scenario.c scenario.h $(GEN_DEPS)

#spacket.obj:   spacket.c spacket.h scc.h $(GEN_DEPS)

stpi.obj:         stpi.c stpi.h print.h tlm.h $(GEN_DEPS)

terms.obj:        terms.c terms.h $(GEN_DEPS)

tlm.obj:          tlm.c tlm.h ad.h bcm.h pcb.h $(GEN_DEPS)

startup.obj:   startup.asm
```

All of the modules are linked together by XLINK86 which uses a file named dcs.lk (shown below) to identify each module.

```
startup.obj, dcs.obj, ad.obj, bcm.obj, clock.obj, edac.obj, eps.obj, gen_apis.obj, modem.obj, msu.obj,
pcb.obj, print.obj, scc.obj, stpi.obj, terms.obj, tlm.obj, &
slibca.ssi&
to dcs.lnk
```

Finally, the linked ROM image needs to have all address relocated to absolute addresses corresponding to the location of the ROM in the embedded system. This process is called loading, and is performed by the program XLOC86 which uses the file dcs.loc (shown below) to describe the relocation needed.

```
dcs.lnk to dcs.abs &
NOINITCODE &
ORDER(CS(&
        FAR_DATA_BEG, FAR_DATA, FAR_DATA_END,&
        FAR_BSS_BEG, FAR_BSS, FAR_BSS_END,&
        HUGE_BSS_BEG, HUGE_BSS, HUGE_BSS_END,&
        DATA_BEG, DATA, CONST, MSG, DATA_END, &
        BSS, BSS_END,&
        STACK,&
        CODE, CODE_END,&
        BOOTSTRAP))&
ADDRESSES(CS(FAR_DATA_BEG(0400H), CODE(0F0000h)))
```

APPENDIX J. SOFTWARE SOURCE CODE

This appendix contains the software source code for the ROM Boot Loader software for the System Controller. The following source code files are included:

Module (filename)	Description	Page(s)
ad.h, ad.c	A/D converter ISR and support routines.	120 - 129
bcm.h, bcm.c	Battery control monitor.	130 - 146
clock.h, clock.c	Clock.	147 - 148
cmd.h, cmd.c	Command interpreter.	149 - 151
dcs.h, dcs.c	main() and master loop for ROM boot loader.	152 - 155
edac.h, edac.c	EDAC ISR and RAM wash.	156 - 158
eps.h, eps.c	EPS support routines	159 - 164
gen_apis.ch gen_apis.c	General (common) subroutines used by other modules.	165 - 167
gen_defs.h	General #defines, typedefs, and macros.	168 - 169
int.h, int.c	CPU interrupt control and support.	170 - 170
modem.h, modem.c	Modem (PA-100) support routines.	171 - 176
msu.h, msu.c	Mass storage support routines.	177 - 192
pcb.h, pcb.c	PCB support routines.	193 - 197
print.h, print.c	Display support for STPI: printf()-like facilities.	198 - 204
rf.h, rf.c	RF support routines.	205 - 207
scc.h, scc.c	SCC support routines.	208 - 217
scenario.h, scenario.c	Scenario (alternate hw/sf) support routines.	218 - 218
spacket.h, spacket.c	Synchrounous packet protocol support routines.	219 - 219
startup.asm	Startup (80186 assembler) module.	220 - 240
stpi.h, stpi.c	Spacecraft test port interface (RS-232) support routines.	241 - 273
terms.h, terms.c	Terminal emulation support for STPI.	274 - 276
tlm.h, tlm.c	Telemetry management support routines.	277 - 283

ad.h ad.c

```
/***************************************************************************
 *
 *    AD.H
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *
 *    Date            Who    What
 *    --------------+-------+------------------------------------------------
 *    15 April 1996   Jah     Creation
 *    25 Feb 1997     Jah     ROM version: reflects EPS port changes
 *
 ***************************************************************************/


/* Approximate time for a A/D Set to be aquired.  This is the approximate
 * difference in time between successive current readings to the batteries.
 */
#define  AD_DELTA_T    1

#define  AD_RES    ((double)(5.0/4095.0))


#ifdef AD
     #define  AD_BASE       0x80
     #define  AD_INSTR0     AD_BASE
     #define  AD_INSTR1     AD_BASE + 2
     #define  AD_INSTR2     AD_BASE + 4
     #define  AD_INSTR3     AD_BASE + 6
     #define  AD_INSTR4     AD_BASE + 8
     #define  AD_INSTR5     AD_BASE + 0x0A
     #define  AD_INSTR6     AD_BASE + 0x0C
     #define  AD_INSTR7     AD_BASE + 0x0E
     #define  AD_CONFIG     AD_BASE + 0x10
     #define  AD_IER        AD_BASE + 0X12
     #define  AD_ISR        AD_BASE + 0x14
     #define  AD_TIMER AD_BASE + 0x16
     #define  AD_FIFO       AD_BASE + 0x18
     #define  AD_LIMIT AD_BASE + 0x1A

     /* Masks */
     #define  RAM00         0x0000
     #define  RAM01         0x0100
     #define  RAM02         0x0200


     /* Define the A/D schedule for readings.
      * The schedule is organized into periods.  For startup, there are
      * 54 periods, number 0 - 53.  Each period uses between one and 5
      * inputs on the A/D.  A/D IN0 is the DCS temperature sensor,
      * A/D IN1 is the Modem temperature sensor, A/D IN2 is the EPS,
      * A/D IN4 is the TMUXA, and A/D IN6 is the TMUXB.
      *
      * Since the EPS contains the current sensors for the batteries and
      * the spacecraft bus (total solar panel current input), and these
      * signals must be read frequently for accurate current integration
      * and quick eclipse sensing, the schedule repeats these readings,
      * interleaving them with other readings.  Each interleaving set
      * is called a Set, numbered from 0 to 13.  There are normally
      * three (3) cycles per set.
      */
     #define  NUM_INPUTS    5
     #define  NUM_PERIODS 42
     #define  NUM_STEPS     14

     #define  NUM_INSTRS    5                     /* Maximum Sequencer instructions */

     #define  NO_PCB        ((unsigned char)0xFF) /* indicates no PCB required for
                                                   * reading; but a reading is needed.
                                                   */
     #define  NO_READ       ((unsigned char)0xFE) /* perform no reading on the A/D
                                                   * channel for a given period.
                                                   */


     /* EPS Setup requires potentially more than one Port 1/2/3 writing
```

```
*    depending on the sensor.  This table is per period and shows
*    for a given period, which port(s) need written to.  Also, this
*    table indicates if the Current Inhibit Switch needs to be
*    disabled, i.e. current reading will be made.
*/
#define  NOT_USED ((unsigned char)0xFF)      /* indicates port is not used */

#define  NO_CUR        ((unsigned char)0x00)     /* not a current measurement */
#define  CURRENT       ((unsigned char)0x01)     /* current: solar bus, solar panels */
#define  CUR_DIR       ((unsigned char)0x02)     /* current w/ directiont */

#define  NO_INSTR ((unsigned int)0xFFFF)     /* instruction not used */


/* This structure describes to the ad_collect() function how to save A/D
 * samples into the correct categories and positions within those categories.
 */
typedef struct ad_collect_params
{
    unsigned char    type;
    unsigned char    position;

} ad_collect_params_struct;

/* Sensor types, used by ad_collect() and ad_collect_params() */
#define  THERMISTOR         ((unsigned char)0)
#define  TEMP_SENSOR        ((unsigned char)1)
#define  V_BATTA            ((unsigned char)2)
#define  V_BATTB            ((unsigned char)3)
#define  V_BUS              ((unsigned char)4)
#define  I_BATTA            ((unsigned char)5)
#define  I_BATTB            ((unsigned char)6)
#define  I_BUS              ((unsigned char)7)
#define  I_SOLAR            ((unsigned char)8)


/* AD Error Flags */
#define  AD_WAIT                  5000
#define  AD_NO_ERROR              0
#define  AD_RESET_ERROR           1
#define  AD_FIFO_COUNT_ERROR      2
#define  AD_CALIB_WAIT_ERROR      3


void            ad_collect(void);
void            ad_init(void);
void interrupt far    ad_isr();
#endif


#ifndef AD
    extern   int                 samples_ready;
    extern   int                 ad_flag;

    extern void                  ad_collect(void);
    extern void                  ad_init(void);
    extern void interrupt far ad_isr();
#endif
```

```
/******************************************************************************
 *
 *    AD.C
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *
 *    Date            Who     What
 *    ---------------+-------+----------------------------------------------
 *    24 April 1996   Jah     Creation
 *    25 Feb 1997     Jah     ROM version includes new EPS port assignments.
 *
 ******************************************************************************/

#include        "gen_defs.h"

#define          AD
#include         "ad.h"
#undef           AD

#include    "bcm.h"
#include    "clock.h"
#include    "eps.h"
#include    "pcb.h"
#include    "tlm.h"


static WORD      period = 0;
static WORD      ad_values[NUM_PERIODS][NUM_INPUTS];
static int       negative_current[NUM_PERIODS];

int samples_ready = FALSE;

int ad_flag = AD_NO_ERROR;



/* Schedule of PCB Commands for EPS reading */
static const BYTE ad_sch_eps[NUM_PERIODS][3] =
{
    /* Current?           Port 3            Port 1 */
    /* Set 0 */
    {CURRENT,            (BYTE)0x80,      (BYTE)0x30}, /* ISC */
    {NO_CUR,      NOT_USED,      (BYTE)0x70}, /* A0 */
    {NO_CUR,      NOT_USED,      (BYTE)0x90}, /* A1 */

    /* Set 1 */
    {CUR_DIR,            (BYTE)0x90,      (BYTE)0x30}, /* IA */
    {NO_CUR,      NOT_USED,      (BYTE)0xF1}, /* A2 */
    {NO_CUR,      NOT_USED,      (BYTE)0xF3}, /* A3 */

    /* Set 2 */
    {CUR_DIR,            (BYTE)0xA0,      (BYTE)0x30}, /* IB */
    {NO_CUR,      NOT_USED,      (BYTE)0xF5}, /* A4 */
    {NO_CUR,      (BYTE)0x01,      (BYTE)0x10}, /* A5 */

    /* Set 3 */
    {CURRENT,            (BYTE)0x80,      (BYTE)0x30}, /* ISC */
    {NO_CUR,      (BYTE)0x03,      (BYTE)0x10}, /* A6 */
    {NO_CUR,      (BYTE)0x05,      (BYTE)0x10}, /* A7 */

    /* (Set 4) */
    {CUR_DIR,            (BYTE)0x90,      (BYTE)0x30}, /* IA */
    {NO_CUR,      (BYTE)0x07,      (BYTE)0x10}, /* A8 */
    {NO_CUR,      NOT_USED,      (BYTE)0xB0}, /* B0 */

    /* (Set 5) */
    {CUR_DIR,            (BYTE)0xA0,      (BYTE)0x30}, /* IB */
    {NO_CUR,      NOT_USED,      (BYTE)0xD0}, /* B1 */
    {NO_CUR,      NOT_USED,      (BYTE)0xF7}, /* B2 */

    /* Set 6 */
    {CURRENT,            (BYTE)0x80,      (BYTE)0x30}, /* ISC */
    {NO_CUR,      NOT_USED,      (BYTE)0xF9}, /* B3 */
    {NO_CUR,      NOT_USED,      (BYTE)0xFB}, /* B4 */

    /* Set 7 */
    {CUR_DIR,            (BYTE)0x90,      (BYTE)0x30}, /* IA */
    {NO_CUR,      (BYTE)0x09,      (BYTE)0x10}, /* B5 */
```

122

```c
    {NO_CUR,            (BYTE) 0x0B,        (BYTE) 0x10},  /* B6 */


    /* Set 8 */
    {CUR_DIR,               (BYTE) 0xA0,        (BYTE) 0x30},  /* IB */
    {NO_CUR,            (BYTE) 0x0D,        (BYTE) 0x10},  /* B7 */
    {NO_CUR,            (BYTE) 0x0F,        (BYTE) 0x10},  /* B8 */


    /* Set 9 */
    {CURRENT,               (BYTE) 0x80,        (BYTE) 0x30},  /* ISC */
    {NO_CUR,            NOT_USED,       (BYTE) 0xFD},  /* VSC */
    {CURRENT,               (BYTE) 0x00,        (BYTE) 0x50},  /* SP0 */


    /* Set 10 */
    {CUR_DIR,               (BYTE) 0x90,        (BYTE) 0x30},  /* IA */
    {CURRENT,               (BYTE) 0x10,        (BYTE) 0x50},  /* SP1 */
    {CURRENT,               (BYTE) 0x20,        (BYTE) 0x50},  /* SP2 */


    /* Set 11 */
    {CUR_DIR,               (BYTE) 0xA0,        (BYTE) 0x30},  /* IB */
    {CURRENT,               (BYTE) 0x30,        (BYTE) 0x50},  /* SP3 */
    {CURRENT,               (BYTE) 0x40,        (BYTE) 0x50},  /* SP4 */


    /* Set 12 */
    {CURRENT,               (BYTE) 0x80,        (BYTE) 0x30},  /* ISC */
    {CURRENT,               (BYTE) 0x50,        (BYTE) 0x50},  /* SP5 */
    {CURRENT,               (BYTE) 0x60,        (BYTE) 0x50},  /* SP6 */


    {CUR_DIR,               (BYTE) 0x90,        (BYTE) 0x30},  /* IA */
    {CUR_DIR,               (BYTE) 0xA0,        (BYTE) 0x30},  /* IB */
    {CURRENT,               (BYTE) 0x70,        (BYTE) 0x50},  /* SP7 */
};


/* Describe how to take the raw samples and organize them by types into
 * the correct sensor type arrays.  This is for the EPS readings only.
 * The other channels are easy to categorize based on the period number
 * (e.g. periods 0 - 31 have TMUX readings, period 0 has DCS & Modem
 * temperature readings).
 */
static const ad_collect_params_struct ad_collect_table[NUM_PERIODS] =
{
    /* Period 0, Set 0 */
    {I_BUS, 0},
    {V_BATTA, 0},
    {V_BATTA, 1},

    /* Period 3, Set 1*/
    {I_BATTA, 0},
    {V_BATTA, 2},
    {V_BATTA, 3},

    /* Period 6, Set 2 */
    {I_BATTB, 0},
    {V_BATTA, 4},
    {V_BATTA, 5},

    /* Period 9, Set 3 */
    {I_BUS, 1},
    {V_BATTA, 6},
    {V_BATTA, 7},

    /* Period 12, Set 4 */
    {I_BATTA, 1},
    {V_BATTA, 8},
    {V_BATTB, 0},

    /* Period 15, Set 5 */
    {I_BATTB, 1},
    {V_BATTB, 1},
    {V_BATTB, 2},

    /* Period 18, Set 6 */
    {I_BUS, 2},
    {V_BATTB, 3},
    {V_BATTB, 4},

    /* Period 21, Set 7 */
    {I_BATTA, 2},
    {V_BATTB, 5},
    {V_BATTB, 6},

    /* Period 24, Set 8 */
    {I_BATTB, 2},
```

123

```
        {V_BATTB, 7},
        {V_BATTB, 8},

        /* Period 27, Set 9 */
        {I_BUS, 3},
        {V_BUS, 0},
        {I_SOLAR, 0},

        /* Period 30, Set 10 */
        {I_BATTA, 3},
        {I_SOLAR, 1},
        {I_SOLAR, 2},

        /* Period 33, Set 11 */
        {I_BATTB, 3},
        {I_SOLAR, 3},
        {I_SOLAR, 4},

        /* Period 36, Set 12 */
        {I_BUS, 4},
        {I_SOLAR, 5},
        {I_SOLAR, 6},

        /* Period 39, Set 13 */
        {I_BATTA, 4},
        {I_BATTB, 4},
        {I_SOLAR, 7},
};


/****************************************************************************
 *
 *   void ad_init()
 *
 *   Initialize the A/D.  Recalibrate.  Setup first (period=0) program.
 *   Set Sequencer Timer (delay before acquisitoin) to 10 msec.
 *
 ****************************************************************************/

void ad_init(void)
{
        int     i;
        int     x;
        WORD temp;


        /* Turn on the TMUXes to allow temperature sensing */
        eps_set_power(PWR_TMUXA, ON);
        pcb_write(TMUXA, 0, 0x10);        /* select channel 0 (calibration resistor) */
        eps_set_power(PWR_TMUXB, ON);
        pcb_write(TMUXB, 0, 0x10);        /* select channel 0 (calibration resistor) */


        /* Zero out values because no A/D has yet occurred (or forcing reset) */
        for (i = 0; i < NUM_PERIODS; i++)
        {
            negative_current[i] = 0;
            ad_values[i][0] = 0;
            ad_values[i][1] = 0;
        }

        period = 0;       /* index into ad_sch[][] */


        /* Setup MUXes A/B for first temperature sensors (Calibration resistors) */
        pcbw_m(TMUXA0, 0, 0x10)
        pcbw_m(TMUXB0, 0, 0x10)

        /* Setup EPS for first reading */
        pcbw_m(EPS0, 3, (ad_sch_eps[0][1]))              /* EPS Port 3 */
        pcbw_m(EPS0, 1, (ad_sch_eps[0][2]))              /* EPS Port 1 */

        eps_set_port2(eps_get_port2());

        outpw(AD_CONFIG, 0x0002);         /* Reset the A/D */
        /* Wait for RESET bit to clear */
        for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0002); x++)
            ;
        if (x == AD_WAIT)
        {
            ad_flag = AD_RESET_ERROR;
            return;
        }
```

124

```c
        outpw(AD_CONFIG, 0x0008);          /* Full Calibration */
        /* Wait for CALIBRATION bit to clear */
        for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0008); x++)
            ;
        if (x == AD_WAIT)
        {
            ad_flag = AD_CALIB_WAIT_ERROR;
            return;
        }

        outpw(AD_CONFIG, 0x0000);            /* stop the sequencer and point to RAM 00 */

        /* Program A/D Sequencer: based on schedule for period 0 */
        outpw(AD_INSTR0, 0xF208);           /* EPS */
        outpw(AD_INSTR1, 0xF208);           /* EPS */
        outpw(AD_INSTR2, 0xF210);           /* TMUX A */
        outpw(AD_INSTR3, 0xF218);           /* TMUX B */
        outpw(AD_INSTR4, 0xF200);           /* DCS */
        outpw(AD_INSTR5, 0xF204);           /* Modem */
        outpw(AD_INSTR6, 0xF202);           /* Pause */


        /* Setup A/D Timer */
        outpw(AD_TIMER, 2000);

        outpw(AD_CONFIG, 0x0002);           /* Reset the A/D */
        /* Wait for RESET bit to clear */
        for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0002); x++)
            ;
        if (x == AD_WAIT)
        {
            ad_flag = AD_RESET_ERROR;
            return;
        }

        /* Force A/D to interrupt when 6 readings in the FIFO occur */
        outpw(AD_IER, 0x3004);

        /* Clear any interrupts of the A/D by reading the status register */
        inpw(AD_ISR);

        /* Allow IRQs from A/D LM12H458 to the CPU */
        #define I1CON    0xFF3A
        outpw(I1CON, 0x0005);               /* Edge-trig., ON, priority 5 */

        /* Start the A/D Sequencer.  Interrupt will occur eventually */
        outpw(AD_CONFIG, 0x0001);     /* start sequencer */

        while (samples_ready == FALSE)      /* wait for end of first A/D sweep */
            ;

}   /* End of ad_init() */


/**********************************************************************************
 *
 *  void interrupt far    ad_isr()
 *
 **********************************************************************************/

void interrupt far    ad_isr()
{
    register int p;
    register int x;
    BYTE          temp;
    static  int   c = 6;        /* first A/D period has six samples to read */


    outpw(AD_CONFIG, 0x0000);    /* Stop sequencer, point to RAM 00 */
    inpw(AD_ISR);                /* this clears the interrupt */

    p = period;       /* get a register copy of the current period */


    /*********************/
    /* Collect A/D samples */
    /*********************/

    /* Wait for all the samples in the FIFO */
    for (x = 0; (x < AD_WAIT) && (((inpw(AD_ISR) & 0xF800) >> 11) < c); x++)
        ;
    if (x == AD_WAIT)
```

125

```
{
    ad_flag = AD_FIFO_COUNT_ERROR;
    /* Send non-specific EOI to Interrupt Controller */
    outpw(0xFF22, 0x8000);

    return;
}

if (p == 0)        /* 0th period has all six A/D samples */
{
    inpw(AD_FIFO);   /* discard - double EPS reading */
    ad_values[0][0] = inpw(AD_FIFO) & 0x0FFF;  /* EPS */
    ad_values[0][1] = inpw(AD_FIFO) & 0x0FFF;  /* TMUXA */
    ad_values[0][2] = inpw(AD_FIFO) & 0x0FFF;  /* TMUXB */
    ad_values[0][3] = inpw(AD_FIFO) & 0x0FFF;  /* DCS */
    ad_values[0][4] = inpw(AD_FIFO) & 0x0FFF;  /* Modem */
}

else if (p < 32) /* 1st - 31st periods have 4 A/D samples */
{
    inpw(AD_FIFO);   /* discard - double EPS reading */
    ad_values[p][0] = inpw(AD_FIFO) & 0x0FFF;  /* EPS */
    ad_values[p][1] = inpw(AD_FIFO) & 0x0FFF;  /* TMUXA */
    ad_values[p][2] = inpw(AD_FIFO) & 0x0FFF;  /* TMUXB */
}

else             /* remaining periods have only 2 A/D sample */
{
    inpw(AD_FIFO);   /* discard - double EPS reading */
    ad_values[p][0] = inpw(AD_FIFO) & 0x0FFF;  /* EPS */
}


/**********************************/
/* Check Current Direction Sensing */
/**********************************/

/* Was there a battery current reading in EPS ? */
if (ad_sch_eps[p][0] == CUR_DIR)
{
    /* Read the direction. */
    pcbr_m(EPS1, 1, temp)        /* Port 5 of the EPS */

    /* Was it Battery A or B ? */
    if (ad_sch_eps[p][1] == 0x90) /* Port 3 tells MUX selection */
        negative_current[p] = (temp & 0x01) ? FALSE : TRUE;
    else
        negative_current[p] = (temp & 0x02) ? FALSE : TRUE;
}


/*********************************/
/* Setup via PCB for new readings */
/*********************************/

/* Has a complete set of data been read?  Ready to start over? */
p++;
if (p >= NUM_PERIODS)
{
    p = 0;
    samples_ready = TRUE;
}
period = p;


/* PCB commands for the MUXes */
if (p < 32)
{
    pcbw_m(TMUXA0, 0, 0x10+p)
    pcbw_m(TMUXB0, 0, 0x10+p)
}

/* PCB commands for the EPS */
if (ad_sch_eps[p][1] != NOT_USED)            /* Program EPS Port 3 ? */
{
    pcbw_m(EPS0, 3, ad_sch_eps[p][1])
}


/* There is always something sent to EPS Port 1 */
pcbw_m(EPS0, 1, ad_sch_eps[p][2])


/*****************************/
```

```
    /* Setup A/D for new readings */
    /*****************************/

    outpw(AD_CONFIG, 0x0002);         /* RESET, point to RAM 00 */

    /* Wait for RESET bit to clear */
    for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0002); x++)
        ;
    if (x == AD_WAIT)
    {
        ad_flag = AD_RESET_ERROR;
        /* Send non-specific EOI to Interrupt Controller */
        outpw(0xFF22, 0x8000);
        return;
    }


    if (p == 0)
    {
        outpw(AD_INSTR0, 0xF208);         /* EPS */
        outpw(AD_INSTR1, 0xF208);         /* EPS */
        outpw(AD_INSTR2, 0xF210);         /* TMUX A */
        outpw(AD_INSTR3, 0xF218);         /* TMUX B */
        outpw(AD_INSTR4, 0xF200);         /* DCS */
        outpw(AD_INSTR5, 0xF204);         /* Modem */
        outpw(AD_INSTR6, 0xF202);         /* Pause */

        outpw(AD_IER, 0x3004);                /* interrupt w/ 6 samples in FIFO */
        c = 6;
    }
    else if (p < 32)
    {
        outpw(AD_INSTR0, 0xF208);         /* EPS */
        outpw(AD_INSTR1, 0xF208);         /* EPS */
        outpw(AD_INSTR2, 0xF210);         /* TMUX A */
        outpw(AD_INSTR3, 0xF218);         /* TMUX B */
        outpw(AD_INSTR4, 0xF202);         /* Pause */
        outpw(AD_IER, 0x2004);                /* interrupt w/ 4 samples in FIFO */
        c = 4;
    }
    else
    {
        outpw(AD_INSTR0, 0xF208);         /* EPS */
        outpw(AD_INSTR1, 0xF208);         /* EPS */
        outpw(AD_INSTR2, 0xF202);         /* Pause */
        outpw(AD_IER, 0x1004);                /* interrupt w/ 2 sample in FIFO */
        c = 2;
    }

    outpw(AD_CONFIG, 0x0008);         /* Full Calibration */
    /* Wait for CALIBRATION bit to clear */
    for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0008); x++)
        ;
    if (x == AD_WAIT)
    {
        ad_flag = AD_CALIB_WAIT_ERROR;

        /* Send non-specific EOI to Interrupt Controller */
        outpw(0xFF22, 0x8000);
        return;
    }

    outpw(AD_CONFIG, 0x0001); /* start sequencer */

    /* Send non-specific EOI to Interrupt Controller */
    outpw(0xFF22, 0x8000);

}   /* End of ad_isr() */


/*****************************************************************************
 *
 *   void ad_check()
 *
 *   Check ad_flag for error condition.
 *
 *****************************************************************************/

void ad_check(void)
{
    static int    reset_count = 0;
    static int    fifo_count  = 0;
    static int    calib_count = 0;
    DWORD         reset_time  = 0L;
```

127

```c
        DWORD           fifo_time   = 0L;
        DWORD           calib_time  = 0L;
        DWORD           t;


        t = get_elapsed_time();
        switch(ad_flag)
        {
            case AD_NO_ERROR:
                break;

            case AD_RESET_ERROR:
                /* This is a bad problem with no real fix.  try initializing again. */
                if ((++reset_count > 10) && (t - calib_time < ONE_MINUTE))
                {
                    /* Force a shutdown */
                }
                ad_init();
                break;

            case AD_FIFO_COUNT_ERROR:
                /* There was a count error trying to access the FIFO in ad_isr().
                 * Try initializing the A/D again and try acquisition.
                 */
                if ((++fifo_count > 10) && (t - calib_time < ONE_MINUTE))
                {
                    /* Force a shutdown */
                }
                ad_init();
                break;

            case AD_CALIB_WAIT_ERROR:
                /* This occurs only within ad_init().  It is a bad problem with no
                 * real fix.
                 */

                /* Force a shutdown */
                break;

            default:
                break;
        }

        ad_flag = AD_NO_ERROR;

}   /* End of ad_check() */


/*****************************************************************************
 *
 *  void ad_collect()
 *
 *  Take all raw sensor readings from the last entire round of A/D collecting
 *  and organize into the telemetry structure within the .sensors structure.
 *
 *  This organization takes data collected in sequence from the A/D ISR, and
 *  places it into the .sensors structure which is organized per sensor type.
 *
 *  This is simplified by using ad_collect_table[] which has an entry for
 *  every A/D acquisition that took place in the last A/D sweep.
 *
 *****************************************************************************/

void ad_collect(void)
{
    int i;
    int pos;


    for (i = 0; i < NUM_PERIODS; i++)
    {
        if (i == 0)
        {
            /* DCS & Modem Temps, EPS, TMUXA, and TMUXB readings are available */
            tlm.sensors.tmp[0] = ad_values[0][3];
            tlm.sensors.tmp[1] = ad_values[0][4];

            /* Read EPS below.... */

            tlm.sensors.ts[0] = ad_values[0][1];   /* TMUXA */
            tlm.sensors.ts[1] = ad_values[0][2];   /* TMUXB */

        }   /* End of IF (i == 0) */
```

128

```
    else if (i < 32)
    {
        /* Read EPS below.... */

        /* EPS, TMUXA, and TMUXB readings are available */
        tlm.sensors.ts[i*2] = ad_values[i][1];      /* TMUXA */
        tlm.sensors.ts[(i*2)+1] = ad_values[i][2]; /* TMUXB */

    }  /* End of IF (i < 32) */

    /* All periods have an EPS reading, move it */
    pos = ad_collect_table[i].position;
    switch(ad_collect_table[i].type)
    {
        case V_BATTA:
            tlm.sensors.vbatta[pos] = ad_values[i][0];
            break;

        case V_BATTB:
            tlm.sensors.vbattb[pos] = ad_values[i][0];
            break;

        case V_BUS:
            tlm.sensors.vscbus = ad_values[i][0];
            break;

        case I_BATTA:
            tlm.sensors.ibatta[pos] = ad_values[i][0];
            if (negative_current[i])
                tlm.sensors.ibatta[pos] |= 0x8000;
            break;

        case I_BATTB:
            tlm.sensors.ibattb[pos] = ad_values[i][0];
            if (negative_current[i])
                tlm.sensors.ibattb[pos] |= 0x8000;
            break;

        case I_BUS:
            tlm.sensors.iscbus[pos] = ad_values[i][0];
            break;

        case I_SOLAR:
            tlm.sensors.isolar[pos] = ad_values[i][0];
            break;

        default: /* ERROR */
            i++;
            i--;
            break;

    }  /* End of SWITCH (ad_collect_table[i].type) */

    }  /* End of FOR */

    samples_ready = FALSE;        /* flagged by ad_main_isr() - waiting for next A/D */

}  /* End of ad_collect() */
```

End of ad.h ad.c

bcm.h bcm.c

```
/***************************************************************************
 *
 *   BCM.H
 *
 *   Battery Charge Monitor for Pansat
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   29 Jan 1997     Jah     Creation
 *
 ***************************************************************************/


#define  BCM_V_TH_STEPS    5         /* # of looksups in the voltage threshold (vs.
                                      * temperature) lookup table
                                      */
#define  BCM_NUM_BATS 2          /* # of batteries */
#define  BCM_NUM_CELLS     9         /* cells/battery */

#define  BCM_NUM_MODES     3         /* Low, Standby, and Normal operation modes */

/* Power Use Modes determined by charge states of the batteries */
#define  BCM_MODE_LOW       0
#define  BCM_MODE_STANDBY 1
#define  BCM_MODE_NORMAL        2


/* Values to define the battery which is online and target (charging). */
#define  BAT_A           0
#define  BAT_B           1
#define  BAT_NONE 2

/* Battery Controls */
#define  CTRL_TRICKLE 0x01
#define  CTRL_ONLINE        0x02
#define  CTRL_DISCHARGE     0x04
#define  CTRL_CHARGE        0x08

/* Operational modes, depending on state of batteries */
#define  MODE_LOW       BCM_MODE_LOW
#define  MODE_STANDBY BCM_MODE_STANDBY
#define  MODE_NORMAL        BCM_MODE_NORMAL


/* This structure holds all of the BCM variables which includes all of the
 * charge state history variables as well as variables which are used to
 * make decisions within the BCM.
 *
 * The following structure is used by other modules to access the BCM states
 * by using the void bcm_info(bcm_info_struct *info) function.
 */
typedef struct bcm_info
{
     int                    count[BCM_NUM_BATS];
     float                  cap[BCM_NUM_BATS];
     int                    online;
     int                    target;
     int                    mode;
     int                    dod_detect;
     int                    eclipse;
     unsigned long int timer;
     int                    online_switch;
     WORD                   control[BCM_NUM_BATS];

     /* reconfigurable variables which affect charging decisions */
     float     dod[BCM_NUM_BATS];
     float     temp_max;
     int       count_max;
     float     over_int;
     float     cap_max[BCM_NUM_BATS];
     float     efficiency[BCM_NUM_BATS];

     float     v_threshold[BCM_NUM_BATS][BCM_V_TH_STEPS];
```

130

```c
        float    vth_low[BCM_NUM_BATS];

        unsigned long intover_times[BCM_NUM_MODES];

}  bcm_info_struct;


/* This structure holds all of the BCM variables which can be modified by other
 * modules when it is desired to change the parameters of the BCM.
 *
 * The following structure is used in conjunction with the BCM function named
 * bcm_set_params(bcm_params_struct *params).
 */
typedef struct bcm_params
{
        int      dod[BCM_NUM_BATS];
        float    temp_dt;
        float    temp_max;
        int      count_max;
        float    over_int;
        float    cap_max[BCM_NUM_BATS];
        float    efficiency[BCM_NUM_BATS];
        float    v_threshold[BCM_NUM_BATS][BCM_V_TH_STEPS];
        float    vth_low[BCM_NUM_BATS];
        unsigned long intover_times[BCM_NUM_MODES];

}  bcm_params_struct;



/* Include specifics for BCM.C */
#ifdef   BCM
    /* Redefine shorthands for BCM routines */
    #define  NUM_BATS BCM_NUM_BATS
    #define  V_TH_STEPS   BCM_V_TH_STEPS
    #define  NUM_MODESBCM_NUM_MODES
    #define  NUM_CELLS    BCM_NUM_CELLS
    #define  NUM_CELL_TEMPS   10


    /* Charge state history */
    #define  CS_BAD           -3
    #define  CS_UNKNOWN        -2
    #define  CS_FORCE_OVER     -1
    #define  CS_OVERCHARGED    0
    /* positive charge state values are counts (the # of re-charge cycles on a battery */

    /* Initial parameters */
    #define  DOD             0.40/* Depth of Discharge (% BELOW 100%) */
    #define  TEMP_LOW     5.0      /* Minimum temperature before heating batteries */
    #define  TEMP_MAX     35.00    /* Maximum temperature for charging */
    #define  COUNT_MAX    5         /* # of REcharges before OVERcharge */
    #define  OVER_INTEGRATE   1.20/* OVER Current integration allowed
                                     * (safety check) before stop charging */
    #define  CAP_A           4.40/* Battery A A*hr capacity */
    #define  CAP_B           4.40/* Battery B A*hr capacity */
    #define  EFF_A           0.83/* Battery A Efficiency */
    #define  EFF_B           0.83/* Battery B Efficiency */

    /* Timers (in seconds) for overcharge checking */
    #define  TIMER_LOW        (2.5L * SECS_PER_HOUR)
    #define  TIMER_STANDBY    (3.5L * SECS_PER_HOUR)
    #define  TIMER_NORMAL (4.5L * SECS_PER_HOUR)

    #define  TRICKLE_TIME FIVE_MINUTES

    /* This is the time set on target_time[] to indicate that a battery has not
     * been charged yet.
     */
    #define  MAX_TARGET_TIME  ((unsigned long int) 4294967296)


    #define  VTH_LOW_BATA 1.20/* Undervoltage threshold to trigger overcharge */
    #define  VTH_LOW_BATB 1.20

    /* Low cell voltages that trigger battery preference decisions */
    #define  VLOW         0.90/* Minimum cell voltage for "healthy" battery */
    #define  VMAX_LOW     1.10/* Minimum Maximum cell voltage for charged battery */


    static void      bcm_charge(void);
    static int       bcm_check_conditions(void);
    static void      bcm_mode(void);
    static int       bcm_in_eclipse(void);
```

131

```
        static void      bcm_online(int preferred);
        static void      bcm_overcharge(void);
        static void      bcm_recharge(void);
        static void      bcm_set_params(bcm_params_struct *params);
        static int       bcm_set_switches(void);
        static void      bcm_target(int preferred);
        static int       bcm_tbound(signed char t);
        static void      bcm_v_max_clear(int battery);
        static float bcm_v_max_min(int battery);
        static float bcm_v_min(int battery);
        void             bcm_tlm_update(void);

#endif


/* Includes for all modules (except BCM.C) referencing this BCM.H */
#ifndef BCM
        extern int   bcm_get_mode(void);
        extern void bcm_info(bcm_info_struct *);
        extern void bcm_init(void);
        extern void  bcm_main(void);
        extern void bcm_set_params(bcm_params_struct *);
        extern void bcm_tlm_update(void);
#endif


/*******************************************************************************
 *
 *    BCM.C
 *
 *    Battery Charge Monitor for Pansat
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *    Date             Who      What
 *    ---------------+-------+---------------------------------------------
 *    29 Jan 1997      Jah      Creation
 *    7 Feb 1997       Jah      Remove Delta-Temperature checks in charging.
 *    24 March 1997    Jah      Use get_elpased_time() instead of get_time().
 *                              Check for trickle charge time for both batteries.
 *
 *******************************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include "gen_defs.h"


#include "clock.h"
#include "pcb.h"

#define      BCM
#include "bcm.h"
#undef       BCM

#include "ad.h"
#include "dcs.h"
#include "eps.h"
#include "tlm.h"


/* Assume unknown charge state for both batteries (i.e. dead) */
static int       count[NUM_BATS]  = {CS_UNKNOWN, CS_UNKNOWN};

/* Assume no capacity for both batteries */
static double    cap[NUM_BATS]    = {0.0, 0.0};

static int       online                    = BAT_NONE;
static int       target                    = BAT_NONE;
static int       mode                  = MODE_LOW;
static int       dod_detect                = FALSE;
static int       eclipse                   = TRUE;
static unsigned long int  over_timer   = 0L;
static int       online_switch             = FALSE;
static WORD      control[BCM_NUM_BATS] = {0, 0};


/* Time since a battery was choosen as a Target.  This is originally set to
 * "infinity", so that the BCM knows that this parameter has not been used.
```
132

```
 * Otherwise, this timer indicates how long a target has been charging.
 * This variable is only used when a battery's charget state history is
 * UNKNOWN (e.g. deployment, reset, or data corruption).
 */
static DWORD      target_time[NUM_BATS] = {0L, 0L};

static float      vbatt_avg[NUM_BATS]  = {0.0, 0.0};    /* Battery Cell Voltage averages */
static signed char   t_avg[NUM_BATS]   = {0.0, 0.0};        /* Battery Cell Temperature averages */

/* Record of growing Maximum cell voltages of a battery while it is charging.
 * This information is used after the battery is charged to determine if
 * the lowest maximum voltage of these cells in a battery are below an
 * acceptable value for a battery which is in good condition.
 */
static float v_max_cells[NUM_BATS][NUM_CELLS] =
{
    {0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0}
};


/* reconfigurable variables which affect charging decisions */
static float dod[NUM_BATS]          = {DOD, DOD};
static float temp_max         = TEMP_MAX;
static int        count_max        = COUNT_MAX;
static float over_int          = OVER_INTEGRATE;
static float cap_max[NUM_BATS]     = {CAP_A, CAP_B};
static float efficiency[NUM_BATS] = {EFF_A, EFF_B};
static float vth_low[NUM_BATS]     = {VTH_LOW_BATA, VTH_LOW_BATB};

#define T_LOW    -5.0
#define T_HIGH    35.0
static float v_threshold[NUM_BATS][V_TH_STEPS] =
{
    {1.46, 1.46, 1.46, 1.46, 1.46},
    {1.46, 1.46, 1.46, 1.46, 1.46}
};

static unsigned long int  over_times[NUM_MODES] =
{
    TIMER_LOW, TIMER_STANDBY, TIMER_NORMAL
};



/*****************************************************************************/
/*****************************************************************************/

/*****************************************************************************
 *
 *   void bcm_main(void)
 *
 *****************************************************************************/

void bcm_main(void)
{
    int preferred;


    /* get new telemetry */
    bcm_tlm_update();

    /* Determine if in eclipse */
    eclipse = bcm_in_eclipse();

    /* Determine preferred battery */
    preferred = bcm_check_conditions();

    bcm_online(preferred);
    bcm_target(preferred);
    bcm_mode();
    bcm_charge();

    if (bcm_on)
        bcm_set_switches();

}  /* End of bcm_main() */


/*****************************************************************************
 *
 *   void bcm_charge(void)
 *
 *   Determine if charging is to occur.
```

133

```
 *
 ***************************************************************************/

void bcm_charge(void)
{
    if (eclipse == FALSE)
    {
        if (target != BAT_NONE)
        {
            /* There is a Target, and the spacecraft is NOT in eclipse.
             * What type of charging does the Target require?
             */
            if ((count[target] < CS_OVERCHARGED) || (count[target] >= count_max))
                bcm_overcharge();

            else
                bcm_recharge();
        }

    }

}  /* End of bcm_charge() */


/*****************************************************************************
 *
 *   int bcm_check_conditions(void)
 *
 *   Determine condition of batteries and wether or not there is a preferred
 *   battery to use.
 *
 ***************************************************************************/

#define WARM          -2
#define COOL          -1
#define NUM_COND_STATES  5

int bcm_check_conditions(void)
{
    int           not_op[NUM_BATS];
    int           preferred;
    int           i, a, b;
    static int    action[NUM_COND_STATES][NUM_COND_STATES] =
            {
                {WARM,    BAT_A,    BAT_A,      BAT_A,    BAT_B},
                {BAT_B,   WARM,     BAT_A,      BAT_B,    BAT_B},
                {BAT_B,   BAT_B,    BAT_NONE,   BAT_B,    BAT_B},
                {BAT_B,   BAT_A,    BAT_A,      COOL,     BAT_B},
                {BAT_A,   BAT_A,    BAT_A,      BAT_A,    COOL}
            };
    static signed char   condition[NUM_COND_STATES] = {-10, 0,   35, 45, MAX_CHAR};


    /* start by assuming both batteries are usable */
    not_op[BAT_A] = not_op[BAT_B] = FALSE;

    /* First, check if there is a target battery */
    if (target != BAT_NONE)
    {
        /* Want to look at the lowest cell voltage, but only after the target
         * has been trickle charged, and then normal charged for the fixed
         * length of times.
         */
        if (get_elapsed_time() > target_time[target] + TRICKLE_TIME)
        {
            /* Make sure lowest cell voltage is above a certain amount. */
            if (bcm_v_min(target) < VLOW)
                not_op[target] = TRUE;
        }
    }


    /* Now, check if any battery has already been charged at least once.  If
     * so, and the battery is NOT the target (being charged), then check the
     * lowest of the maximum cell voltages while the battery was charging.  This
     * minimum of the maximum cell voltages must be above a certain amount.
     */
    for (i = 0; i < NUM_BATS; i++)
    {
        if (i != target)
        {
            if (count[i] != CS_UNKNOWN)
            {
                /* Check minimum of the maximum cell voltages */
```

134

```c
                if (bcm_v_max_min(i) < VMAX_LOW)
                    not_op[i] = TRUE;
            }
        }
    }


    /* If there is a preferred battery, then indicate.  Otherwise, do temperature
     * tests to still see if there is a preferred battery.
     */
    preferred = not_op[BAT_A] ? (not_op[BAT_B] ? BAT_NONE : BAT_B) : (not_op[BAT_B] ? BAT_A : BAT_NONE);

    if (preferred == BAT_NONE)
    {
        /* For each battery, find which temperature category it falls within */
        for (i = 0; i <= NUM_COND_STATES - 1; i++)
        {
            if (t_avg[BAT_A] < condition[i])
                break;
        }
        a = i;

        for (i = 0; i <= NUM_COND_STATES - 1; i++)
        {
            if (t_avg[BAT_B] < condition[i])
                break;
        }
        b = i;

        switch(action[b][a])
        {
            case WARM:
                preferred = (t_avg[BAT_A] >= t_avg[BAT_B]) ? BAT_A : BAT_B;
                break;

            case COOL:
                preferred = (t_avg[BAT_A] <= t_avg[BAT_B]) ? BAT_A : BAT_B;
                break;

            case BAT_A:
                preferred = BAT_B;
                break;

            case BAT_B:
                preferred = BAT_B;
                break;

            case BAT_NONE:
                preferred = BAT_NONE; /* no preference */
                break;

            default: /* error */
                preferred = BAT_NONE;
        }
    }

    return(preferred);

}  /* End of bcm_check_conditions() */


/****************************************************************************
 *
 *   int bcm_get_mode(void)
 *
 *   Report which operation mode the batteries are capable of supporting.
 *
 ****************************************************************************/

int bcm_get_mode(void)
{
    return(mode);

}  /* End of bcm_get_mode() */


/****************************************************************************
 *
 *   int bcm_in_eclipse(void)
 *
 *   Determine if the spacecraft is in eclipse.
 *
 ****************************************************************************/
```

135

```c
int bcm_in_eclipse(void)
{
    if (tlm_cnv.iscbus < 0.050)
        return(TRUE);
    else
        return(FALSE);

}   /* End of bcm_in_eclipse */


/*******************************************************************************
 *
 *   void bcm_info(*bcm_info_struct info)
 *
 *   Return information that describes the state of the BCM.
 *
 *******************************************************************************/

void bcm_info(bcm_info_struct * info)
{
    register int i, j;


    info->online    = online;
    info->target    = target;
    info->mode         = mode;
    info->dod_detect = dod_detect;
    info->eclipse       = eclipse;

    /* This is for the elased overcharge timer for the target battery */
    if (over_timer != 0L)
        info->timer = get_elapsed_time() - over_timer; /* duration of overcharging */
    else
        info->timer = 0L;                           /* indicate not in use */


    info->online_switch  = online_switch;

    info->temp_max   = temp_max;
    info->count_max  = count_max;
    info->over_int   = over_int;

    for (i = 0; i < NUM_BATS; i++)
    {
        info->count[i]    = count[i];
        info->cap[i] = cap[i];

        info->control[i] = control[i];

        info->cap_max[i] = cap_max[i];
        info->efficiency[i]   = efficiency[i];

        info->vth_low[i] = vth_low[i];

        info->dod[i] = dod[i];

        for (j = 0; j < V_TH_STEPS; j++)
            info->v_threshold[i][j] = v_threshold[i][j];
    }

    for (i = 0; i < NUM_MODES; i++)
        info->over_times[i] = over_times[i];

}   /* End of bcm_info() */


/*******************************************************************************
 *
 *   void bcm_init(void)
 *
 *   Initialize the BCM. This is used to force a reset of the BCM for example
 *   when a data error has occurred and the charge state variables are no
 *   not valid.
 *
 *******************************************************************************/

void bcm_init(void)
{
    register int i, j;
    int             preferred;


    /* Determine preferred battery */
    preferred = bcm_check_conditions();
```

136

```c
    if (preferred != BAT_NONE)
    {
        /* There is a preferred battery due to environmental/battery
         * performance criteria.  Use this battery regardless of
         * other charge state history information.
         */
        online = preferred;
    }
    else
        online = BAT_A;        /* otherwise, default to Battery A */

    target          = BAT_NONE;
    mode            = MODE_LOW;
    dod_detect      = FALSE;
    eclipse         = TRUE;
    over_timer          = 0L;
    online_switch   = FALSE;

    temp_max        = TEMP_MAX;
    count_max           = COUNT_MAX;
    over_int        = OVER_INTEGRATE;

    cap_max[0]          = CAP_A;
    cap_max[1]          = CAP_B;

    efficiency[0]   = EFF_A;
    efficiency[1]   = EFF_B;

    vth_low[0]          = VTH_LOW_BATA;
    vth_low[1]          = VTH_LOW_BATB;


    for (i = 0; i < NUM_BATS; i++)
    {
        /* Maximum cell voltages, recorded while battery is charging. */
        for (j = 0; j < NUM_CELLS; j++)
            v_max_cells[i][j] = 0.0;

        count[i] = CS_UNKNOWN;
        cap[i] = 0.0;
        target_time[i] = 4294967295L;

        t_avg[i] = 0.0;

        dod[i] = DOD;
    }

    over_times[MODE_LOW]        = TIMER_LOW;
    over_times[MODE_STANDBY]    = TIMER_STANDBY;
    over_times[MODE_NORMAL]         = TIMER_NORMAL;

    /* Turn OFF all battery controls, but leave the ONLINE battery online */
    eps_batts_off(online);

}   /* End of bcm_init() */


/*****************************************************************************
 *
 *   int bcm_mode(void)
 *
 *   Determine which operation mode the batteries are capable of supporting.
 *
 *****************************************************************************/

void bcm_mode(void)
{
    register int other;


    other = (online == BAT_A) ? BAT_B : BAT_A;

    if (count[online] == CS_UNKNOWN)
        mode = MODE_LOW;

    else if (count[other] == CS_UNKNOWN)
        mode = MODE_LOW;

    else if (cap[online] < cap_max[online]*(1.0 - dod[online]))
        mode = (cap_max[other] < cap_max[other]*(1.0 - dod[other])) ? MODE_LOW : MODE_STANDBY;

    else
        mode = (cap[online] < cap_max[online]*(1.0 - dod[online])) ? MODE_STANDBY : MODE_NORMAL;
```

137

```c
}  /* End of bcm_mode() */


/***************************************************************************
 *
 *   void bcm_online(void)
 *
 *   Determine which battery should be online.
 *
 ***************************************************************************/

void bcm_online(int preferred)
{
    online_switch = FALSE;


    if (preferred != BAT_NONE)
    {
        /* There is a preferred battery due to environmental/battery
         * performance criteria.  Use this battery regardless of
         * other charge state history information.
         */
        if (preferred != online)
            online_switch = TRUE;

        online = preferred;

        if ( (cap[online] <= cap_max[online]*(1.0 - dod[online])) ||
             (vbatt_avg[online] < vth_low[online]) )
            dod_detect = TRUE;

        return;
    }


    /* Is there already a battery online ? */
    else if (online != BAT_NONE)
    {
        /* Has the voltage of the online battery dropped below its
         * low voltage threshold.
         */
        if ((vbatt_avg[online] < vth_low[online]) && (count[online] >= 0))
        {
            /* The online battery has charge history and is not
             * acknowledge as needing OVERCharge.  Yet, its average
             * voltage has dropped below a voltage threshold, signifying
             * it has less capacity than expected.  Therefore, force it to
             * be overcharged in its next charge cycle.
             */
            count[online] = CS_FORCE_OVER;
        }

        /* Is the online battery below DOD ? */
        if ( (cap[online] <= cap_max[online]*(1.0 - dod[online])) ||
             (vbatt_avg[online] < vth_low[online]) )
        {
            /* What is the Target?  If there is already a Target, then
             * the Target must remain charging, and the Online battery must
             * remain online until the Target finishes charging.  Therefore,
             * DOD has been reached and a temperature reference must be set
             * on the Target since charging will now occur at a quicker rate.
             */
            if (target != BAT_NONE)
            {
                /* There is a Target battery.  Is this the first time this
                 * (below DOD) been detected? If so, mark the Target's
                 * temperature for charging decisions.
                 */
                if (!dod_detect)
                    dod_detect = TRUE;
            }

            else/* There is no Target, switch the online battery */
            {
                online_switch = TRUE;
                switch(online)
                {
                    case BAT_A:
                        online = BAT_B;
                        break;
                    case BAT_B:
                        online = BAT_A;
                        break;
                    default:
```

138

```
                              /* This is an ERROR case and should never happen */
                              online = (cap[BAT_A] >= cap[BAT_B]) ? BAT_A : BAT_B;
                              break;
                    }
               }

          } /* End of cap[online]... */

          else/* enough capacity still, no action */
          {
          }

     } /* End of if (online != BAT_NONE) */


     else/* There is no online battery, choose one */
     {
          if (count[BAT_A] == CS_UNKNOWN)
               /* Battery A has no charge state history, how about Battery B? */
               online = (count[BAT_B] == CS_UNKNOWN) ? BAT_A : BAT_B;

          else/* A has charge state history, but what about B ? */
          {
               if (count[BAT_B] == CS_UNKNOWN)
                    online = BAT_A;
               else
                    online = cap[BAT_A] >= cap[BAT_B] ? BAT_A : BAT_B;
          }

          /* Since a new battery has JUST been selected to become online, force
           * DOD detect to FALSE.
           */
          dod_detect = FALSE;

     } /* End of else (no online battery...) */

} /* End of bcm_online() */


/*****************************************************************************
 *
 *   void bcm_overcharge(void)
 *
 *   Perform charging on the Target using the OVERcharge method.
 *
 *****************************************************************************/

void bcm_overcharge(void)
{
     /* First, check to make sure the temperature of the Target is low enough. */
     /* However, this ignored when first starting in case there was a hot-soak
      * on the spacecraft while in the shuttle bay which would cause the
      * spacecraft to be hot when ejected.....immediate cooling is expected.
      */
     if ((t_avg[target] <= temp_max) || (get_elapsed_time() < ONE_HOUR))
     {
          /* Check so that the Target has not OVER Integrated, a safety check */
          if (cap[target] < cap_max[target]*over_int)
          {
               /* Has the Target reached the voltage threshold which is indicative
                * over beginning to reach the overcharge portion of the charge
                * cycle?  If overcharge timer (overtime) is NOT zero, then it is
                * already active, which means the voltage threshold was already reached
                * earlier.
                */
               if ( (vbatt_avg[target] < (v_threshold[target][bcm_tbound((signed char)t_avg[target])])
                    && (over_timer == (unsigned long int)0L)) )
               {
                    /* Below the voltage threshold, continue charging */
                    /* no new action needed. */
               }

               else/* OVERcharge */
               {
                    if (over_timer == 0L)
                    {
                         /* Begin the overcharge, mark the time.  Non-zero means the timer
                          * is active.
                          */
                         over_timer = get_elapsed_time();
                    }

                    else/* Already in overcharge, how is it going? */
                    {
```
139

```c
                    if (get_elapsed_time() >= (over_timer + over_times[mode]))
                    {
                        count[target] = CS_OVERCHARGED;
                        cap[target] = cap_max[target];
                        target = BAT_NONE;
                        dod_detect = FALSE;
                        over_timer = 0L;
                    }

                    else
                    {
                        /* Continue overcharge using NON DOD detect method. */
                        /* No new action needed. */
                    }
                }

            }

        }

        else/* OVER integration occured, stop the charging. */
        {
            count[target] = CS_OVERCHARGED;
            cap[target] = cap_max[target];
            target = BAT_NONE;
            dod_detect = FALSE;
        }

    }

    else/* TOO HOT */
    {
        /* Do not change capacity; however, it is ASSUMED that the battery charging
         * is not complete.  So, the count is placed to CS_FORCE_OVER, so that
         * the next time this battery becomes the Target, it will be forced into
         * OVERcharge.
         */
        count[target] = CS_FORCE_OVER;
        target = BAT_NONE;
        dod_detect = FALSE;
    }

}  /* End of bcm_overcharge() */


/****************************************************************************
 *
 *   void bcm_recharge(void)
 *
 *   Perform charging on the Target using the REcharge method.
 *
 ****************************************************************************/

void bcm_recharge(void)
{
    /* First, check to make sure the temperature of the Target is low enough. */
    /* However, this ignored when first starting in case there was a hot-soak
     * on the spacecraft while in the shuttle bay which would cause the
     * spacecraft to be hot when ejected.....immediate cooling is expected.
     */
    if ((t_avg[target] <= temp_max) || (get_elapsed_time() < ONE_HOUR))
    {
        if (cap[target] >= cap_max[target])
        {
            /* REcharged */
            count[target]++;
            cap[target] = cap_max[target];
            dod_detect = FALSE;
            target = BAT_NONE;
        }

        else
        {
            /* Continue recharge.  No new action needed. */
        }
    }

    else/* TOO HOT */
    {
        /* Do not change capacity; however, it is ASSUMED that the battery charging
         * is not complete.  So, the count remains the same as well.
         */
        target = BAT_NONE;
        dod_detect = FALSE;
```

140

```
        }

}   /* End if bcm_recharge() */


/*****************************************************************************
 *
 *   void bcm_set_params(bcm_params_struct *params)
 *
 *   Allow BCM parameters to be modified.
 *
 *****************************************************************************/

void bcm_set_params(bcm_params_struct *params)
{
    register int i, j;


    temp_max = params->temp_max;
    count_max = params->count_max;
    over_int = params->over_int;

    for (i = 0; i < NUM_BATS; i++)
    {
        cap_max[i] = params->cap_max[i];
        efficiency[i] = params->efficiency[i];
        vth_low[i] = params->vth_low[i];

        for (j = 0; j < V_TH_STEPS; j++)
                v_threshold[i][j] = params->v_threshold[i][j];

        dod[i] = params->dod[i];
    }

    for (i = 0; i < NUM_MODES; i++)
        over_times[i] = params->over_times[i];

} /* End of bcm_set_params() */


/*****************************************************************************
 *
 *   void bcm_set_switches()
 *
 *   Set the battery switches according to all of the decisions made within one
 *   pass through the BCM.
 *
 *****************************************************************************/

int bcm_set_switches(void)
{
    register int offline;
    register int not_target;
    int             mode;


    /* Indicate which controls are set.  Begin by setting all control status off */
    offline = (online == BAT_A) ? BAT_B : BAT_A;
    control[online] = 0;
    control[offline] = 0;

    not_target = (target == BAT_A) ? BAT_B : BAT_A;


    /* Turn on ONLINE switch for battery to be online.  It doesn't matter if
     * the online bateries are being switched (A->B) or (B->A) as long as you
     * FIRST turn ONLINE a battery, and THEN turn OFFLINE the other.
     */
    if (eps_set_battery(online, BAT_ONLINE) == ERROR)
        return(ERROR);
    if (eps_set_battery(offline, BAT_OFFLINE) == ERROR)
        return(ERROR);
    control[online] |= CTRL_ONLINE;


    if (eclipse)            /* turn OFF battery controls, EXCEPT online */
        eps_batts_off(online);

    else if (target != BAT_NONE)  /* turn ON all that needs to be - except online (already done) */
    {
        /* CHARGING: Normal or Trickle ? */

        /* Only do Trickle charging on a battery that has no charge state
         * history, and that has only been selected to be charged within
```

141

```
        * the TRICKLE_TIME time frame.
        */
       if ((count[target] == CS_UNKNOWN) &&
           (get_elapsed_time() - target_time[target]) < TRICKLE_TIME)
       {
           /* Make sure all normal charging switches are OFF */
           if (eps_set_battery(not_target, BAT_CHARGE_OFF) == ERROR)
               return(ERROR);
           if (eps_set_battery(target, BAT_CHARGE_OFF) == ERROR)
               return(ERROR);

           if (eps_set_battery(not_target, BAT_TRICKLE_OFF) == ERROR)
               return(ERROR);
           if (eps_set_battery(target, BAT_TRICKLE_ON) == ERROR)
               return(ERROR);

           control[target] |= CTRL_TRICKLE;

       }  /* End of IF (Trickle Charging) */

       else/* Time for Normal Charging */
       {
           /* Make sure all trickle charging switches are OFF */
           if (eps_set_battery(not_target, BAT_TRICKLE_OFF) == ERROR)
               return(ERROR);
           if (eps_set_battery(target, BAT_TRICKLE_OFF) == ERROR)
               return(ERROR);

           if (eps_set_battery(not_target, BAT_CHARGE_OFF) == ERROR)
               return(ERROR);
           if (eps_set_battery(target, BAT_CHARGE_ON) == ERROR)
               return(ERROR);

           control[target] |= CTRL_CHARGE;

       }  /* END of ELSE (Normal Charging) */

   }  /* End of ELSE  (target != BAT_NONE) */

   else    /* There is NO Target, so make sure all controls are OFF */
   {
       eps_batts_off(online);        /* turn OFF battery controls, except online */
   }


   /* Now, check for the battery heaters */
   mode = (t_avg[BAT_A] < TEMP_LOW) ? ON : OFF;
   eps_set_power(PWR_HEATA, mode);

   mode = (t_avg[BAT_B] < TEMP_LOW) ? ON : OFF;
   eps_set_power(PWR_HEATB, mode);

}  /* End of bcm_set_switches() */


/*****************************************************************************
 *
 *   void bcm_target(void)
 *
 *   Determine which battery should be (targeted) to charge, if any.
 *
 *****************************************************************************/

void bcm_target(int preferred)
{
   if (preferred != BAT_NONE)
   {
       /* There is a preferred battery to set as the target due to
        * environmental and battery performance criteria.
        */

       /* Is there a target battery already, and is it the same as the
        * preferred?  If so, do not interrupt (restart) the target
        * selection process.
        */
       if (preferred != target)
       {
           if (count[preferred] == CS_UNKNOWN)
           {
               target = preferred;
               cap[target] = 0.0;
           }

           else if (count[preferred] == CS_FORCE_OVER)
```

142

```c
                target = preferred;

        else if (cap[preferred] <  cap_max[preferred]*(1.0 - dod[preferred]))
                target = preferred;

        /* else, the preferred battery is not ready to charge - leave it */

        if (target != BAT_NONE)
        {
            /* zero out the cell voltage maxes */
            bcm_v_max_clear(target);

            over_timer = 0L;        /* indicate just determined the target */
                            /* this is the overcharge timer */

            target_time[target] = get_elapsed_time();

        }
    }
}


else if (target == BAT_NONE)
{
    /* There is no target, try to choose one if appropriate. */
    if (count[BAT_A] == CS_UNKNOWN)
    {
        /* A has no charge state history, it is the Target; however,
         * if B is also the same, mark its charge history likewise.
         */
        if (count[BAT_B] == CS_UNKNOWN)
            cap[BAT_B] = 0.0;

        target = BAT_A;
        cap[BAT_A] = 0.0;
    }

    /* How about B ? */
    else if (count[BAT_B] == CS_UNKNOWN)
    {
        /* Battery B has no charge state history, but A does. */
        target = BAT_B;
        cap[BAT_B] = 0.0;
    }

    else
    {
        /* Does A need to be forced to overcharge ? */
        if (count[BAT_A] == CS_FORCE_OVER)
            target = BAT_A;

        /* A does not need a overcharge forced, how about B? */
        else if (count[BAT_B] == CS_FORCE_OVER)
            target = BAT_B;

        else/* Need to look at Capacities now in order to decide. */
        {
            if (cap[BAT_A] <= cap[BAT_B])
                target = (cap[BAT_A] <= cap_max[BAT_A]*(1.0 - dod[BAT_A])) ? BAT_A : BAT_NONE;

            else
                target = (cap[BAT_B] <= cap_max[BAT_B]*(1.0 - dod[BAT_B])) ? BAT_B : BAT_NONE;
        }
    }

    if (target != BAT_NONE)
    {
        /* zero out the cell voltage maxes */
        bcm_v_max_clear(target);

        over_timer = 0L;      /* indicate just determined the target */
                        /* this is the overcharge timer */

        target_time[target] = get_elapsed_time();

    }

}  /* End of IF (target == BAT_NONE) */


else
{
    /* The Target has ALREADY been previously choosen, leave it. */
}
```

```c
}   /* End of bcm_target() */


/**************************************************************************
 *
 *   int bcm_tbound(float t)
 *
 *   This funcion categorizies a temperature value into an index
 *   for table lookup into the voltage vs. temperature table
 *
 *       v_threshold[target][temperature]
 *
 *   to check if a
 *   charging battery has reached the voltage threshold.
 *
 *   Jah: optimize
 *
 **************************************************************************/

int bcm_tbound(signed char t)
{
    unsigned char range   = (unsigned char)(T_HIGH - T_LOW);
    unsigned char dt = (unsigned char)(range/V_TH_STEPS);

    signed char      acc = (signed char)T_LOW; /* accumulator from T_LOW to T_HIGH in steps of dt */
    register int i    = 0;                          /* index corresponding to the temperature */


    if (t >= T_HIGH)
        return(V_TH_STEPS - 1);

    else
    {
        while (t > acc)
        {
            i++;
            acc += dt;
        }

        return(i);
    }

}   /* End of bcm_tbound() */


/**************************************************************************
 *
 *   void bcm_tlm_update()
 *
 **************************************************************************/

#define  BAT_A_TS0    2
#define  BAT_B_TS0    12

void bcm_tlm_update(void)
{
    unsigned register int i, j;
    WORD                  a, b;
    WORD                  ta, tb;
    int                   na, nb;
    static  DWORD          t_old = 0;        /* for dTime for Cap. calcs */


    /* Voltages */
    vbatt_avg[BAT_A] = tlm_cnv.vcellsa_avg;
    vbatt_avg[BAT_B] = tlm_cnv.vcellsb_avg;


    /* Now, update the maximum cell voltages */
    for (j = 0; j < NUM_CELLS; j++)
        if (v_max_cells[BAT_A][j] < tlm_cnv.vcellsa[j])
            v_max_cells[BAT_A][j] = tlm_cnv.vcellsa[j];
    for (j = 0; j < NUM_CELLS; j++)
        if (v_max_cells[BAT_B][j] < tlm_cnv.vcellsb[j])
            v_max_cells[BAT_B][j] = tlm_cnv.vcellsb[j];

    /* Capacities */
    if (tlm_cnv.ibatta < -0.020)
        cap[BAT_A] += tlm_cnv.ibatta*((double)(tlm_record.etime - t_old)/(double)SECS_PER_HOUR);
    else if (tlm_cnv.ibatta > 0.020)
        cap[BAT_A] += efficiency[BAT_A]*tlm_cnv.ibatta*((double)(tlm_record.etime -
t_old)/(double)SECS_PER_HOUR);
```

```c
    if (tlm_cnv.ibattb < -0.020)
        cap[BAT_B] += tlm_cnv.ibattb*((double)(tlm_record.etime - t_old)/(double)SECS_PER_HOUR);
    else if (tlm_cnv.ibattb > 0.020)
        cap[BAT_B] += efficiency[BAT_B]*tlm_cnv.ibattb*((double)(tlm_record.etime -
t_old)/(double)SECS_PER_HOUR);
    t_old = tlm_record.etime;


    /* Temperatures */
    /* Get first average with ALL measurements */
    for (ta = 0, tb = 0, i = 0; i < NUM_CELL_TEMPS; i++)
    {
        ta += tlm_cnv.ts[BAT_A_TS0 + i];
        tb += tlm_cnv.ts[BAT_B_TS0 + i];
    }
    a = ta/NUM_CELL_TEMPS;
    b = tb/NUM_CELL_TEMPS;
    /* Now, remove any measurements that are above/below 5 degrees */
    for (na = 10, nb = 10, i = 0; i < NUM_CELL_TEMPS; i++)
    {
        if ((tlm_cnv.ts[BAT_A_TS0 + i] >= (a + 5)) ||
            (tlm_cnv.ts[BAT_A_TS0 + i] <= (a - 5)))
        {
            ta -= tlm_cnv.ts[BAT_A_TS0 + i];
            na--;
        }

        if ((tlm_cnv.ts[BAT_B_TS0 + i] >= (b + 5)) ||
            (tlm_cnv.ts[BAT_B_TS0 + i] <= (b - 5)))
        {
            tb -= tlm_cnv.ts[BAT_B_TS0 + i];
            nb--;
        }
    }
    /* Recompute the average */
    if (na == 0)
        t_avg[BAT_A] = a;      /* weird case - avoid divide by zero */
    else
        t_avg[BAT_A] = ta/na;

    if (nb == 0)
        t_avg[BAT_B] = b;      /* weird case - avoid divide by zero */
    else
        t_avg[BAT_B] = tb/nb;

}  /* End of bcm_tlm_update() */


/***************************************************************************
 *
 *   void bcm_v_max_clear(int battery)
 *
 *   Set the maximum cell voltages history all to zero in order to restart the
 *   recording of maximum cell voltages for a particular battery.
 *
 ***************************************************************************/

void bcm_v_max_clear(int battery)
{
    register int i;


    for (i = 0; i < NUM_CELLS; i++)
        v_max_cells[battery][i] = 0.0;

}  /* End of bcm_v_max_clear() */


/***************************************************************************
 *
 *   void bcm_v_max_min(int battery)
 *
 *   Return the minimum cell voltage of all of the recorded maximum cell
 *   voltages for a particular battery.
 *
 ***************************************************************************/

float   bcm_v_max_min(int battery)
{
    register int i;
    float   x = 100.0;


    for (i = 0; i < NUM_CELLS; i++)
```

145

```
            if (v_max_cells[battery][i] < x)
                x = v_max_cells[battery][i];

    return(x);

}  /* End of bcm_v_max_min() */


/****************************************************************************
 *
 *   void bcm_v_min(int battery)
 *
 *   Return the minimum cell voltage from the current telemetry record of cell
 *   voltages for a particular battery.
 *
 ****************************************************************************/

float    bcm_v_min(int battery)
{
    register int i;
    float    x = 100.0;


    if (battery == BAT_A)
        for (i = 0; i < NUM_CELLS; i++)
            if (tlm_cnv.vcellsa[i] < x)
                x = tlm_cnv.vcellsa[i];

    else  /* Must be battery B */
        for (i = 0; i < NUM_CELLS; i++)
            if (tlm_cnv.vcellsb[i] < x)
                x = tlm_cnv.vcellsb[i];

    return(x);

}  /* End of bcm_v_min() */
```

End of bcm.h bcm.c

clock.h, clock.c

```
/***************************************************************************
 *
 *   CLOCK.H
 *
 *   Real time clock for PANSAT
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   8 March 1996 Jah      Creation
 *
 ***************************************************************************/

/* Tick intervals for day and time stamping */
#define  SECS_PER_MIN 60L
#define  SECS_PER_HOUR    (SECS_PER_MIN*60L)
#define  SECS_PER_DAY (SECS_PER_HOUR*24L)
#define  SECS_PER_YEAR    (SECS_PER_DAY*365L)

#define  THIRTY_SECONDS   (30L)
#define  ONE_MINUTE       (1L*SECS_PER_MIN)             '           '
#define  TWO_MINUTES      (2L*SECS_PER_MIN)
#define  FIVE_MINUTES (5L*SECS_PER_MIN)
#define  TEN_MINUTES      (10L*SECS_PER_MIN)
#define  ONE_HOUR      (1L*SECS_PER_HOUR)

#ifdef   CLOCK
     /* Timer2 is programmed to interrupt once every 1/60 second */
     #define  TICKS_PER_SECOND 60

     void interrupt far    clock_isr();
     DWORD                 get_elapsed_time(void);
     DWORD                 get_time(void);
     void             set_time(DWORD);

#endif


#ifndef  CLOCK
     extern void interrupt far clock_isr();
     extern DWORD             get_elapsed_time(void);
     extern DWORD             get_time(void);
     extern void                 set_time(DWORD);
#endif

/***************************************************************************
 *
 *   CLOCK.C
 *
 *   Real time clock for PANSAT
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   8 March 1996 Jah      Adoption from STAR's clock.c
 *
 ***************************************************************************/

#include    "gen_defs.h"
#include "gen_apis.h"

#define      CLOCK
#include "clock.h"
#undef       CLOCK

#include "edac.h"

static DWORD sec_count = 0L;            /* total elapsed from time of startup */
```

```c
static DWORD base_time = 0L;          /* UTC offset uses sec_count */

DWORD           icount = 0L;

/****************************************************************************
 *
 *   clock_isr()
 *
 *   This routine is invoked by Timer 2 (INT 13h) - the system clock tick
 *   mechanism.  tick_count is the total number of ticks since the real time
 *   clock was initialized.
 *
 ****************************************************************************/

void     interrupt far clock_isr()
{
    static unsigned int interval = 0;


    icount++;

    if (++interval == TICKS_PER_SECOND)
    {
        interval = 0;
        sec_count++;
        /* edac_ram_wash(); */
    }

    /* Send non-specific EOI to Interrupt Controller */
    outpw(0xFF22, 0x8000);

}   /* End of clock_isr() */

/****************************************************************************
 *
 *   DWORD get_elapsed_time(void)
 *
 *   Return system up time in number of elapsed seconds.
 *
 ****************************************************************************/

DWORD get_elapsed_time(void)
{

    return(sec_count);

}   /* End of get_elapsed_time() */

/****************************************************************************
 *
 *   DWORD get_time(void)
 *
 *   Return system time in number of elapsed seconds.  Based on 1 Jan 1970.
 *
 ****************************************************************************/

DWORD get_time(void)
{
    return(base_time + sec_count);

}   /* End of get_time() */


/****************************************************************************
 *
 *   void     set_time(DWORD t)
 *
 *   base_time maintains the date/time at which the clock was set to a specific
 *   time.  Thus, base_time + sec_count is the current time.  While sec_count
 *   maintains the elapsed time since startup.
 *
 ****************************************************************************/

void     set_time(DWORD t)
{
    base_time = t - sec_count;

}   /* End of set_time() */
```

End of clock.h, clock.c

cmd.h, cmd.c

```
/****************************************************************************
 *
 *   CMD.H
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *
 ***************************************************************************/


/****************************************************************************
 *
 *   CMD.C
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *   19 Nov 1996     Jah     Creation
 *
 ***************************************************************************/

#include     "gen_defs.h"

#define       CMD
#include "cmd.h"
#undef        CMD


typedef struct ack_packet
{
    BYTE cmd;
    BYTE action;

} acl_packet_struct;

typedef struct  cmd_packet
{
    BYTE cmd;


}

/* Upcoming Command Types (from grounstation to spacecraft) */
#define  CMD_CONFIRM       0x55
#define  CMD_CONTROL       0x5A
#define  CMD_EXEC      0xA5
#define  CMD_GETP      0xAA
#define  CMD_LOAD      0x66
#define  CMD_MAP           0x69
#define  CMD_RESET         0x96
#define  CMD_SETP      0x99
#define  CMD_STATUS        0x00
#define  CMD_SLOG_CLEAR    0x0F
#define  CMD_SLOG_READ     0xF0
#define  CMD_VERIFY        0xFF
#define  CMD_UNKNOWN       0x56     /* not an actual command, but used in the
                                     * spacecraft ACK packet to identify the
                                     * type of command that was received.
                                     */

/* Downgoing Action Types (from spacecraft to groundstation) */
#define  ACT_CONFIRM       0x55
#define  ACT_NONE      0xAA
#define  ACT_REPEAT        0x66
#define  ACT_UNKNOWN       0x99
```

149

```
/*******************************************************************************
 *
 *   cmd_examine()
 *
 *******************************************************************************/

cmd_examine()
{

    switch(in_packet->cmd)
    {
        case CMD_CONFIRM:
            ack_packet->cmd = CMD_CONFIRM;
            ack_packet->action = ACT_CONFIRM;
            break;

        case CMD_CONTROL:
            ack_packet->cmd = CMD_CONTROL;
            ack_packet->action = ACT_CONFIRM;
            break;

        case CMD_EXEC:
            ack_packet->cmd = CMD_EXEC;
            ack_packet->action = ACT_CONFIRM;
            break;

        case CMD_GETP:
            ack_packet->cmd = CMD_GETP;
            ack_packet->action = ACT_NONE;
            break;

        case CMD_LOAD:
            ack_packet->cmd = CMD_LOAD;
            ack_packet->action = ACT_NONE;
            break;

        case CMD_MAP:
            ack_packet->cmd = CMD_MAP;
            ack_packet->action = ACT_NONE;
            break;

        case CMD_RESET:
            ack_packet->cmd = CMD_RESET;
            ack_packet->action = ACT_CONFIRM;
            break;

        case CMD_SETP:
            ack_packet->cmd = CMD_SETP;
            ack_packet->action = ACT_CONFIRM;
            break;

        case CMD_STATUS:
            ack_packet->cmd = CMD_STATUS;
            ack_packet->action = ACT_NONE;
            /* append telemetry */
            break;

        case CMD__SLOG_CLEAR:
            ack_packet->cmd = CMD_SLOG_CLEAR;
            ack_packet->action = ACT_CONFIRM;
            /* delete all stored telemetry */
            break;

        case CMD_SLOG_READ:
            ack_packet->cmd = CMD_SLOG_READ;
            ack_packet->action = ACT_NONE;
            /* append first record of stored telemetry */
            break;

        case CMD_VERIFY:
            ack_packet->cmd = CMD_VERIFY;

            ack_packet->action = ACT_NONE;

            ack_packet->action = ACT_REPEAT;
            break;

        default: /* UNKNOWN */
            /* Valid packet (CRC passed), but command code is not valid */
            ack_packet->cmd = CMD_UNKNOWN;
            ack_packet->action = ACT_UNKNOWN;
            break;
    }
```

```
        /* Place the command in the history buffer */

        /* Now, send the packet to the packet driver */
        scc_send_packet(out_packet);

}   /* End of cmd_examine() */
```

End of cmd.h, cmd.c

dcs.h, dcs.c

```
/*****************************************************************************
 *
 *    DCS.H
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *
 *    Date              Who     What
 *    ---------------+-------+----------------------------------------------
 *    2 Nov 1993             Jah            Creation
 *
 *****************************************************************************/

#ifdef    DCS

#endif

#ifndef DCS
     extern int debug;
     extern int bcm_on;
#endif

/*****************************************************************************
 *
 *    DCS.C
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *    Date              Who     What
 *    ---------------+-------+----------------------------------------------
 *    2 Nov 1996       Jah     Creation
 *
 *****************************************************************************/

#include    <stdlib.h>

#include    "gen_defs.h"

#define      DCS
/* #include   "dcs.h" */
#undef       DCS

#include    "ad.h"
#include    "bcm.h"
#include    "clock.h"
#include    "eps.h"
#include    "pcb.h"
#include      "print.h"
#include    "scc.h"
#include    "stpi.h"
#include    "terms.h"
#include    "tlm.h"


/* Function prototypes */
static void      boot_loader(void);
static void      boot_loader_setup(void);
static void      info_screen(void);
static void      stpi_only(void);
static void      stpi_only_setup(void);
static int       use_stpi(int *check);


int debug = FALSE;
int bcm_on = TRUE;


/*****************************************************************************
 *
 *    void main(void)
 *
```

```
 *   This is the first routine of the C runtime which is called by the startup
 *   (assembler) module.  This routine is responsible for determining if the
 *   Boot Loader should begin autonomous operations of the spacecraft or if
 *   the STPI is being used and control should be passed to the monitor for
 *   ground-like operations.
 *
 *   To accomplish this, the PANSAT banner is sent out the asynch serial port
 *   in the event that a ground computer is attached and will send a response
 *   back to the spacecraft.  Then this module waits upto 30 seconds for a
 *   response via the ground computer.  If there is one, control is transferred
 *   to the STPI module, by-passing the Boot Loader.  Otherwise, control is
 *   transferred to the Boot Loader which begins autonomous operation of the
 *   spacecraft.
 *
 *   A two way flag is used to check for STPI use (and disabling of the Boot
 *   Loader) so that it is unlikely that just a single flag gets its value
 *   changed accidentally.
 *
 ********************************************************************************/

void main(void)
{
    int stpi_on = FALSE;
    int stpi_check = 0;


    /* Send PANSAT banner to the STPI */
    info_screen();

    /* Jah - check time */
    while (get_elapsed_time() < 5)
    {
        stpi_on = use_stpi(&stpi_check);
        if (stpi_on && (stpi_check == 0x5A))
        {
            stpi_only();
            /* STPI Monitor to be used, ground station computer is ready . */
        }

        /* If STPI Monitor was being used but control is transferred here,
         * then the command to begin the Boot Loader was given.  Thus, let
         * the flow of control go to the WHILE loop below which starts the
         * Boot Loader.
         */
    }


    /* Spacecraft is to run autonomously */
    while (TRUE)
    {
        boot_loader();

        /* The Boot Loader subroutine will run forever unless a command is
         * given to disable the Boot Loader, in which case control will be
         * transferred to the statements below.  Thus, assume the STPI Monitor
         * is to be run.
         */

        /* STPI Monitor */
        stpi_only();

        /* The STPI Monitor subroutine will run forever unless a command is
         * given to begin the Boot Loader.  In this case, transfer of
         * control will be given back to this WHILE loop, a new iteration
         * of the loop will begin, and automatically the Boot Loader
         * will be called.
         */
    }

}  /* End of main() */


/*******************************************************************************
 *
 *   void boot_loader()
 *
 ********************************************************************************/

void boot_loader(void)
{
    static int run_boot_loader = TRUE;


    /* Setup */
```
153

```c
/*  boot_loader_setup(); */


    while (run_boot_loader)
    {
        check_tlm();

        bcm_main();

        /* RF Listen */

        /* Check/Process command (via RF link) */

        /* RF Transmit */

        monitor();

        /* Scenarios */

        /* Reset Watchdog timer */
    }

}   /* End of boot_loader() */


/*****************************************************************************
 *
 *   voidboot_loader_setup()
 *
 *****************************************************************************/

void boot_loader_setup(void)
{
    ad_init();

    if(msu_check_flash_tlm() == ERROR)
    {
        /* There was an error trying to locate an empty record */
    }

    while (!samples_ready)
        ;

    /* make sure BCM charge state variables are setup correctly */
    bcm_main();

    /* setup time to listen timer */

    /* setup communications */

}   /* End of boot_loader_setup() */


/*****************************************************************************
 *
 *       info_screen()
 *
 *****************************************************************************/

void    info_screen(void)
{
    home();
    clr();

    dprint("\nPANSAT System Controller: ");
    dprint("%s\n", __DATE__);
    dprint("80C186 running at 7.3728 MHz\n");
    dprint("ROM :   F000:0 - F000:FFFF (64 k)\n");
    dprint("RAM :   0000:0 - 7000:FFFF (512 k)\n");
    dprint("SCCA:   (Cmd = 2, Data = 6) Synchronous\n");
    dprint("SCCB:   (Cmd = 0, Data = 4) UART at 19.2 kbits/sec, 8N1\n");

    serial_out(CTRL_W);

}   /* End of info_screen() */


/*****************************************************************************
 *
 *   voidstpi_only()
 *
 *****************************************************************************/
```

```c
void stpi_only(void)
{
    static int run_stpi_only = TRUE;


    while (run_stpi_only)
    {
        check_tlm();

        monitor();

    }

}  /* End of stpi_only() */


/*****************************************************************************
 *
 *   void stpi_only_setup()
 *
 *****************************************************************************/
void stpi_only_setup(void)
{
    /* bcm_off(); */

    pcb_write(EPS0, 0, 0);    /* Battery A control, TMUXA, HEATA */
    pcb_write(EPS0, 2, 0);    /* Other subsystem power */
    pcb_write(EPS1, 2, 0);    /* Battery B control */

}  /* End of stpi_only_setup() */


/*****************************************************************************
 *
 *   int   use_stpi()
 *
 *   Monitor the STPI for the sequence of characters that spell PANSAT.  If
 *   these are sensed in this order, then it is assumed that a ground computer
 *   is attached to PANSAT and PANSAT is to go into the STPI monitor mode.
 *
 *   This routine is called repetitively during the first 30 seconds and thus
 *   keeps a record of any received characters from the SPTI and when six are
 *   received does a compare to the character sequence "PANSAT".
 *
 *****************************************************************************/

int use_stpi(int *check)
{
    static char  check_chars[6];       /* 6 chars for PANSAT */
    static int   c = 0;                /* index into check_chars[] */


    if (is_serial_in())
        check_chars[c++] = serial_in();

    if (c == 6)
    {
        if (strncmp(check_chars, "PANSAT", 6) == 0)
        {
            *check = 0x5A;
            serial_out(0x5A);
            return(TRUE);
        }
    }

    return(FALSE);

} /* End of use_stpi() */
```

End of dcs.h, dcs.c

edac.h, edac.c

```
/*****************************************************************************
 *
 *    EDAC.H
 *
 *    Error Detection And Correction routines.
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *
 *    Revision History:
 *    =================
 *
 *    Date            Who     What
 *    ---------------+-------+-----------------------------------------------
 *    21 August 1996   Jah      Creation
 *
 *****************************************************************************/


typedef struct edac_stats
{
    BYTE huge *  ram_wash_ptr;
    DWORD        soft_errors;
    DWORD        ram_wash_cycles;

} edac_stats_struct;


#ifdef   EDAC

    void edac_get_stats(edac_stats_struct * stats);
    void    interrupt far edac_hard_isr();
    void interrupt far edac_soft_isr();
    void ram_wash(void);

#endif


#ifndef EDAC

    extern void                    edac_get_stats(edac_stats_struct * stats);
    extern void  interrupt far    edac_hard_isr();
    extern void  interrupt far    edac_soft_isr();
    extern void                    ram_wash(void);

#endif


/*****************************************************************************
 *
 *    EDAC.C
 *
 *    Error Detection And Correction routines.
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *
 *    Revision History
 *    =================
 *    Date            Who     What
 *    ---------------+-------+-----------------------------------------------
 *    21 August 1996   Jah      Creation
 *
 *****************************************************************************/

#include      "gen_defs.h"


#define      EDAC
#include "edac.h"
#undef       EDAC

#include "int.h"
```

```c
#define     D0CON              0xFF00
#define     DMA0_OFF      0xFFFF
#define     DMA1_OFF      0xFFFF
#define     RAM_WASH_COUNT   64              /* 64 words */


/* This marks the location of the RAM wash in Static RAM.  It is updated
 * after ram_wash() finishes an interval of washing so that it marks
 * where next to begin washing the next time ram_wash() is invoked.
 */
static BYTE huge *    ram_wash_ptr = (BYTE huge *)0;

static DWORD edac_hard_err = 0L;
static DWORD edac_soft_err = 0L;
static DWORD ram_wash_cycles = 0L;



/**************************************************************************
 *
 *   void edac_get_stats()
 *
 *   Returns the information about the EDAC.  The current RAM Wash pointer,
 *   the number of soft errors.
 *
 **************************************************************************/

void edac_get_stats(edac_stats_struct * stats)
{
    stats->ram_wash_ptr = ram_wash_ptr;
    stats->soft_errors = edac_soft_err;
    stats->ram_wash_cycles = ram_wash_cycles;

}   /* End of edac_get_stats() */


/**************************************************************************
 *
 *       edac_hard_isr()
 *
 **************************************************************************/

void    interrupt far edac_hard_isr()
{

    /* Send specific EOI to PIC */
    outpw(INT_EOI, 0x000E);        /* Interrupt 0xE (14) - INT 2 */

}   /* End of edac_hard_isr() */


/**************************************************************************
 *
 *       edac_soft_isr()
 *
 **************************************************************************/


void    interrupt far edac_soft_isr()
{
    edac_soft_err++;

    /* Clear the EDAC error */
    pcb_portc(2, RESET);
    pcb_portc(2, SET);

    /* Send specific EOI to PIC */
    outpw(INT_EOI, 0x000F);        /* Interrupt 0xF (15) - INT 3 */

}   /* End of edac_soft_isr() */



/**************************************************************************
 *
 *   void edac_ram_wash(void)
 *
 *   Washes a 128 byte contiguous block of SRAM.  Uses ram_wash_ptr as the
 *   starting address.  Updates ram_wash_ptr upon terminating.
 *
 **************************************************************************/

void edac_ram_wash(void)
{
    _asm
```

```
    {
        push ds

        lds     si, DWORD PTR ram_wash_ptr
        mov     cx, RAM_WASH_COUNT
        cld

        mov     dx, D0CON
        in      ax, dx
        mov     bx, ax                  ; save a copy of DMA0 config
        and     ax, DMA0_OFF
        out     dx, ax                  ; DMA channel 0 disabled

        inc     dx
        inc     dx
        in      ax, dx
        mov     di, ax                  ; save a copy of DMA1 config
        and     ax, DMA1_OFF
        out     dx, ax                  ; DMA channel 1 disabled

        cli
        rep     lodsw
        sti

        mov     ax, di                  ; DMA1 config.
        out     dx, ax                  ; DMA channel 1 enabled
        dec     dx
        dec     dx
        mov     ax, bx                  ; DMA0 config.
        out     dx, ax                  ; DMA channel 0 enabled

        pop     ds
    }

    ram_wash_ptr += 128;                /* 64 words = 128 bytes were washed */
    if (ram_wash_ptr > (BYTE huge *)0x7FFFFFFF)
    {
        ram_wash_ptr = (BYTE huge *)0;
        ram_wash_cycles++;
    }

}   /* End of edac_ram_wash() */
```

End of edac.h, edac.c

eps.h, eps.c

```
/**************************************************************************
 *
 *  EPS.H
 *
 *  Defines for the RF unit interface routines.
 *
 *  Petite Amateur Navy Satellite (PANSAT).
 *  Embedded ROM software.
 *  Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *  Jim A. Horning (Jah)
 *
 *  Revision History:
 *  =================
 *  Date              Who      What
 *  ---------------+-------+------------------------------------------
 *  30 Oct 1996       Jah      Creation
 *
 **************************************************************************/


/* Power control defines */
#define  PWR_TMUXA     0
#define  PWR_MSA       1
#define  PWR_HEATA     2
#define  PWR_HEATB     3
#define  PWR_RF        4
#define  PWR_TMUXB     5
#define  PWR_MSB       6
#define  PWR_ANTREL    7

/* Battery Control defines */
#define  BAT_ONLINE          1
#define  BAT_OFFLINE         2
#define  BAT_TRICKLE_ON      3
#define  BAT_TRICKLE_OFF     4
#define  BAT_CHARGE_ON       5
#define  BAT_CHARGE_OFF      6
#define  BAT_DISCHARGE_ON 7
#define  BAT_DISCHARGE_OFF 8



#ifdef EPS
    #define  POWER_ON_DELAY    0x1FFF

    void eps_batts_off(int);
    void eps_global_off(void);
    void     eps_set_port2(BYTE value);
    BYTE eps_get_port2(void);
    BYTE eps_get_battery(void);
    int      eps_set_battery(int battery, int mode);
    WORD     eps_get_power(void);
    void     eps_set_power(int device, int mode);
    void eps_reset_wdog(void);

#endif


#ifndef EPS
    extern void       eps_batts_off(int);
    extern void       eps_global_off(void);
    extern void       eps_set_port2(BYTE value);
    extern BYTE       eps_get_port2(void);
    extern BYTE       eps_get_battery(void);
    extern int        eps_set_battery(int battery, int mode);
    extern WORD       eps_get_power(void);
    extern void       eps_set_power(int device, int mode);
    extern void       eps_reset_wdog(void);

#endif
```

159

```
/**************************************************************************
 *
 *   EPS.C
 *
 *   Interface routines for the EPS unit.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+-------------------------------------------------
 *   30 Oct 1996     Jah     Creation
 *   6 Dec 1996      Jah     Changed port settings per Ron's EPS modifications
 *   25 Feb 1997     Jah     Keep Port 2, Bit 1 always on (the current MUX enable)
 *
 **************************************************************************/

#include     "gen_defs.h"

#define       EPS
#include "eps.h"
#undef        EPS

#include "bcm.h"
#include "pcb.h"


int check_bat(int battery, BYTE mask);


static BYTE  port0 = 0;
static BYTE port2 = 0x01;      /* Current MUX enabled */
static BYTE port6 = 0;


/**************************************************************************
 *
 *   eps_batts_off()
 *
 *   Power OFF all battery controls, EXCEPT the battery ONLINE controls.
 *
 **************************************************************************/

void eps_batts_off(int online)
{
    if (online == BAT_A)
    {
        port0 &= 0x0F;         /* ALL Battery A controls to be turned OFF */
        port0 |= 0x20;         /* Battery A ONLINE to be turned ON */
        port6 = 0;             /* ALL Battery B controls to be turned OFF */

        /* Write to port0 FIRST to insure there remains a battery online */
        pcb_write(EPS0, 0, port0);
        pcb_write(EPS1, 2, port6);
    }

    else if (online == BAT_B)
    {
        /* Turn OFF All battery controls to A */
        port0 &= 0x0F.         /* ALL battery A controls to be turned OFF */
        port6 = 0x20.          /* Battery B controls to be turned OFF, except ONLINE = ON */

        /* Write to Port6 FIRST to insure there remains a battery online */
        pcb_write(EPS1, 2, port6);
        pcb_write(EPS0, 0, port0);
    }

}   /* End of eps_batts_off() */


/**************************************************************************
 *
 *   eps_global_off()
 *
 *   Power OFF all subsystems, and all battery controls, EXCEPT the battery
 *   ONLINE controls.
 *
 **************************************************************************/

void    eps_global_off(void)
```

160

```
    {
        port0 &= 0x20;
        pcb_write(EPS0, 0, port0);     /* All OFF except the online (if it is on) */

        port2 = 0x01;
        pcb_write(EPS0, 2, port2);     /* Other subsystem power */

        port6 &= 0x20;
        pcb_write(EPS1, 2, port6);     /* Battery B control OFF except the online (if it is on) */

    }   /* End of eps_global_off() */


/*****************************************************************************
 *
 *   eps_set_port2()
 *
 *   Power ON or OFF a subsystem, leaving others undisturbed.
 *
 *****************************************************************************/

void    eps_set_port2(BYTE value)
{
    port2 = value | 0x01;          /* Keep Current MUX enabled */
    pcb_write(EPS0, 2, port2);

}   /* End of eps_set_port2() */


/*****************************************************************************
 *
 *   eps_get_port2()
 *
 *   Power ON or OFF a subsystem, leaving others undisturbed.
 *
 *****************************************************************************/

BYTE eps_get_port2(void)
{
    return(port2);

}   /* End of eps_get_port2() */


/*****************************************************************************
 *
 *       eps_get_power()
 *
 *       Get power ON or OFF status for the subsystems.
 *
 *****************************************************************************/

WORD    eps_get_power(void)
{
    /* LSB is power bits of Port 0, MSB is power bits of Port 2
     * Other bits pertaining to battery control, unused bits, or the
     * S/P current inhibit and strobe are maksed off to 0.
     */

    return( (((WORD)port2 & 0x007C) << 8) | ((WORD)port0 & 0x000D));

}   /* End of eps_get_power() */


/*****************************************************************************
 *
 *   eps_set_power()
 *
 *   Power ON or OFF a subsystem, leaving others undisturbed.
 *
 *****************************************************************************/

void    eps_set_power(int select, int mode)
{
    register BYTE    temp, mask;
    WORD            delay;


    /* This table contains bit positions in EPS ports 0 & 2 for power control
     * bits for the subsystems, heaters, and antenna release.
     */
    static WORD     power_table[] = {4, 8, 1, 0x40, 0x20, 0x10, 8, 4};

    switch(select)
```

161

```
    {
            /* Port 0 Power Controls */
            case PWR_TMUXA:
            case PWR_MSA:
            case PWR_HEATA:
                if (mode == ON)
                    port0 |= power_table[select];
                else if (mode == OFF)
                    port0 &= ~power_table[select];
                else
                    return;
                pcb_write(EPS0, 0, port0);
                break;

            /* Port 2 Power Controls */
            case PWR_HEATB:
            case PWR_RF:
            case PWR_TMUXB:
            case PWR_MSB:
            case PWR_ANTREL:
                if (mode == ON)
                    port2 |= power_table[select];
                else if (mode == OFF)
                    port2 &= ~power_table[select];
                else
                    return;
                pcb_write(EPS0, 2, port2);
                break;

            default:
                break;
    }

    if (mode == ON)
        for (delay = 0; delay < POWER_ON_DELAY; delay++)
            ;

} /* End of eps_set_power() */


/*******************************************************************
 *
 *   eps_get_battery()
 *
 *******************************************************************/

BYTE eps_get_battery(void)
{
    /* MSnibble is Battery A contorl bits of Port 0, LSnibble is Battery B
     * control bits of Port 6.  Other bits pertaining to battery control,
     * unused, or the S/P current inhibit and strobe are maksed off to 0.
     */

    return(((port6 & 0xF0)>>4) | (port0 & 0xF0));

} /* End of eps_get_battery() */


/*******************************************************************
 *
 *   eps_set_battery()
 *
 *******************************************************************/

#define MASK_AND     1
#define MASK_OR      2

int eps_set_battery(int battery, int mode)
{
    register BYTE    temp, mask;


    switch(mode)
    {
        case BAT_CHARGE_ON:
            temp = 0x80; mask = MASK_OR; break;
        case BAT_CHARGE_OFF:
            temp = ~0x80; mask = MASK_AND; break;

        case BAT_DISCHARGE_ON:
            temp = 0x40; mask = MASK_OR; break;
        case BAT_DISCHARGE_OFF:
            temp = ~0x40; mask = MASK_AND; break;
```

162

```
            case BAT_ONLINE:
                temp = 0x20; mask = MASK_OR; break;
            case BAT_OFFLINE:
                temp = ~0x20; mask = MASK_AND; break;

            case BAT_TRICKLE_ON:
                temp = 0x10; mask = MASK_OR; break;
            case BAT_TRICKLE_OFF:
                temp = ~0x10; mask = MASK_AND; break;

            default:
                break;
        }

        switch(battery)
        {
            case BAT_A:
                if ((mask == MASK_OR) && check_bat(BAT_A, temp))
                    port0 |= temp;
                else if (mask == MASK_AND)
                    port0 &= temp;
                else
                {
                    dprint("EPS: battery control command error - state not allowed\n");
                    return(ERROR);
                }
                pcb_write(EPS0, 0, port0);
                break;

            case BAT_B:
                if ((mask == MASK_OR) && check_bat(BAT_B, temp))
                    port6 |= temp;
                else if (mask == MASK_AND)
                    port6 &= temp;
                else
                {
                    dprint("EPS: battery control command error - state not allowed\n");
                    return(ERROR);
                }
                pcb_write(EPS1, 2, port6);
                break;

            default:
                break;
        }

        return(NO_ERROR);

    }   /* End of eps_set_battery() */


/*****************************************************************************
 *
 *   check_bat()
 *
 *   Check to make sure that the new battery control desired to be turned
 *   on does NOT conflict with other settings that are already on.
 *
 *****************************************************************************/

int check_bat(int battery, BYTE mask)
{
    /* Allowed battery ON setting compared to existing settings */
    /* This table is read with the row being the battery ON setting
     * that is desired.  The first four entries are for battery A, the
     * following four are for battery B.  The columns represent
     * settings already set ON for battery control; again, the first four
     * are for battery A, the remaining for for battery B.
     * If there is a 1, then the new setting is NOT allowed.  A zero indicates
     * the new setting is allowed.
     *
     *
     *                      B Trickle    ------------------------+
     *                                                           |
     *                      B Online ---------------------+    |
     *                                                     |    |
     *                      B Discharge ------------------+   |    |
     *                                                     |   |    |
     *                      B Charge     ---------------+  |   |    |
     *                                                 |  |  |   |
     *                      A Trickle    ------------+  |  |  |   |
     *                                             |  |  |  |  |
     *                      A Online ---------+    |  |  |  |  |
     *                                       |   |  |  |  |  |
     *                      A Discharge ------+  |  |  |  |  |  |
```
163

```
 *
 *                        A Charge ---+   |  |  |  |  |  |  |  |
 *                                        |  |  |  |  |  |  |  |
 *                                        V  V  V  V  V  V  V  V
 *
 *                        (hex)         80 40 20 10 08 04 02 01
 */
        static WORD       bat_table[8]= {
                          /* A Charge    */0x59,
                          /* A Discharge */0xB4,
                          /* A Online    */0x40,
                          /* A Trickle   */0xC9,
                          /* B Charge    */0x95,
                          /* B Discharge */0x4B,
                          /* B Online    */0x04,
                          /* B Trickle   */0x9C};

        int     i;


        switch(mask)
        {
            case 0x10:   /* Trickle */
                i = 3;  break;
            case 0x20:   /* Online */
                i = 2;  break;
            case 0x40:   /* Discharge */
                i = 1;  break;
            case 0x80:   /* Charge */
                i = 0;  break;
        }  /* End of SWITCH */

        if (battery == BAT_B)
            i += 4;

        /* i is the index into the table */
        /* The table ANDed with the current information regarding which battery
         * switches are set (ON) are used to see if the new ON request is
         * allowed.
         */
        if (bat_table[i] & eps_get_battery())
            return(FALSE);        /* not allowed */
        else
            return(TRUE);

}  /* End of check_bat() */


/*****************************************************************************
 *
 *    eps_reset_wdog()
 *
 *    Toggle the EPS Watch Dog timer so that the EPS will not reset the current
 *    active System Controller.  This is done using Port 4 of the EPS.
 *
 *****************************************************************************/

void eps_reset_wdog(void)
{
    WORD i;


    pcb_write(EPS1, 0, 1);

    for (i = 0; i < 0x1FFF; i++)
        ;

    pcb_write(EPS1, 0, 0);

}  /* End of eps_reset_wdog() */
```

End of eps.h, eps.c

gen_apis.h, gen_apis.c

```
/***************************************************************************
 *
 *   GEN_APIS.H
 *
 *   Include file for general functions.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *   5 Sept 1996     Jah     Creation
 *
 ***************************************************************************/


#ifdef   GEN_APIS
    WORD crc_calc(WORD crc, BYTE data);
    WORD check_crc(void *ptr, int size);
    WORD disable_ints(void);
    void enable_ints(void);
    WORD prepare_crc(void *ptr, int size);

#endif

/* prototypes for modules other than msu.c */
#ifndef GEN_APIS
    extern WORD   crc_calc(WORD crc, BYTE data);
    extern WORD   check_crc(void *ptr, int size);
    extern WORD   disable_ints(void);
    extern void   enable_ints(void);
    extern WORD   prepare_crc(void *ptr, int size);

    extern const WORD crc_table[];


#endif


/***************************************************************************
 *
 *   GEN_APIS.C
 *
 *   General functions available to .c files..
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *   21 Oct 1996     Jah     Creation
 *
 ***************************************************************************/

#include         "gen_defs.h"

#define           GEN_APIS
    #include "gen_apis.h"
#undef            GEN_APIS


/* CRC lookup table for all 256 CRC combinations from an 8-bit value */
#define  CRC_TABLE_LEN      256
static const WORD      crc_table[CRC_TABLE_LEN] =
{
    0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
    0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
    0x1891, 0x0918, 0x3B83, 0x2A0A, 0x5EB5, 0x4F3C, 0x7DA7, 0x6C2E,
    0x94D9, 0x8550, 0xB7CB, 0xA642, 0xD2FD, 0xC374, 0xF1EF, 0xE066,
    0x3122, 0x20AB, 0x1230, 0x03B9, 0x7706, 0x668F, 0x5414, 0x459D,
    0xBD6A, 0xACE3, 0x9E78, 0x8FF1, 0xFB4E, 0xEAC7, 0xD85C, 0xC9D5,
```

```
    0x29B3, 0x383A, 0x0AA1, 0x1B28, 0x6F97, 0x7E1E, 0x4C85, 0x5D0C,
    0xA5FB, 0xB472, 0x86E9, 0x9760, 0xE3DF, 0xF256, 0xC0CD, 0xD144,
    0x6244, 0x73CD, 0x4156, 0x50DF, 0x2460, 0x35E9, 0x0772, 0x16FB,
    0xEE0C, 0xFF85, 0xCD1E, 0xDC97, 0xA828, 0xB9A1, 0x8B3A, 0x9AB3,
    0x7AD5, 0x6B5C, 0x59C7, 0x484E, 0x3CF1, 0x2D78, 0x1FE3, 0x0E6A,
    0xF69D, 0xE714, 0xD58F, 0xC406, 0xB0B9, 0xA130, 0x93AB, 0x8222,
    0x5366, 0x42EF, 0x7074, 0x61FD, 0x1542, 0x04CB, 0x3650, 0x27D9,
    0xDF2E, 0xCEA7, 0xFC3C, 0xEDB5, 0x990A, 0x8883, 0xBA18, 0xAB91,
    0x4BF7, 0x5A7E, 0x68E5, 0x796C, 0x0DD3, 0x1C5A, 0x2EC1, 0x3F48,
    0xC7BF, 0xD636, 0xE4AD, 0xF524, 0x819B, 0x9012, 0xA289, 0xB300,
    0xC488, 0xD501, 0xE79A, 0xF613, 0x82AC, 0x9325, 0xA1BE, 0xB037,
    0x48C0, 0x5949, 0x6BD2, 0x7A5B, 0x0EE4, 0x1F6D, 0x2DF6, 0x3C7F,
    0xDC19, 0xCD90, 0xFF0B, 0xEE82, 0x9A3D, 0x8BB4, 0xB92F, 0xA8A6,
    0x5051, 0x41D8, 0x7343, 0x62CA, 0x1675, 0x07FC, 0x3567, 0x24EE,
    0xF5AA, 0xE423, 0xD6B8, 0xC731, 0xB38E, 0xA207, 0x909C, 0x8115,
    0x79E2, 0x686B, 0x5AF0, 0x4B79, 0x3FC6, 0x2E4F, 0x1CD4, 0x0D5D,
    0xED3B, 0xFCB2, 0xCE29, 0xDFA0, 0xAB1F, 0xBA96, 0x880D, 0x9984,
    0x6173, 0x70FA, 0x4261, 0x53E8, 0x2757, 0x36DE, 0x0445, 0x15CC,
    0xA6CC, 0xB745, 0x85DE, 0x9457, 0xE0E8, 0xF161, 0xC3FA, 0xD273,
    0x2A84, 0x3B0D, 0x0996, 0x181F, 0x6CA0, 0x7D29, 0x4FB2, 0x5E3B,
    0xBE5D, 0xAFD4, 0x9D4F, 0x8CC6, 0xF879, 0xE9F0, 0xDB6B, 0xCAE2,
    0x3215, 0x239C, 0x1107, 0x008E, 0x7431, 0x65B8, 0x5723, 0x46AA,
    0x97EE, 0x8667, 0xB4FC, 0xA575, 0xD1CA, 0xC043, 0xF2D8, 0xE351,
    0x1BA6, 0x0A2F, 0x38B4, 0x293D, 0x5D82, 0x4C0B, 0x7E90, 0x6F19,
    0x8F7F, 0x9EF6, 0xAC6D, 0xBDE4, 0xC95B, 0xD8D2, 0xEA49, 0xFBC0,
    0x0337, 0x12BE, 0x2025, 0x31AC, 0x4513, 0x549A, 0x6601, 0x7788
};


static WORD ints_disabled = FALSE;


/****************************************************************************
 *
 *   WORD crc_calc(WORD crc, BYTE data)
 *
 *   Takes a CRC and a new data byte and computes a new CRC.
 *
 ****************************************************************************/

WORD crc_calc(WORD crc, BYTE data)
{
    return((crc >> 8) ^ crc_table[data ^ (crc & 0xFF)]);

}  /* End of crc_calc() */


/****************************************************************************
 *
 *   WORD check_crc(void *ptr, int size)
 *
 *   Check an arbitrary lengthed buffer with its CRC.  This function assumes
 *   that the buffer has the CRC appended to the data.  The CRC is returned.
 *
 ****************************************************************************/

WORD check_crc(void *ptr, int size)
{
    int     i;
    WORD crc;
    BYTE *  p = (BYTE *)ptr;


    for (i = 0, crc = 0; i < size; i++)
        crc = crc_calc(crc, *p++);

    return(crc);

}  /* End of prep_crc() */


/****************************************************************************
 *
 *   WORD prepare_crc(void *data, int size)
 *
 *   Prepare an arbitrary lengthed buffer with its CRC by calculating the CRC
 *   and then appending it to the end of the buffer.  This function assumes
 *   that the buffer is prepared with a length of 2 greater than the data.
 *   The high-byte of the CRC is written first, the low-byte follows.  The CRC
 *   is returned.
 *
 ****************************************************************************/

WORD prepare_crc(void *ptr, int size)
```

166

```
{
    int     i;
    WORD crc;
    BYTE *p = (BYTE *)ptr;


    for (i = 0, crc = 0; i < size; i++)
        crc = crc_calc(crc, *p++);

    *p++ = (BYTE)((crc & 0xFF00) >> 8);
    *p = (BYTE)(crc & 0x00FF);

    return(crc);

}   /* End of prep_crc() */


/*******************************************************************************
 *
 *   WORD disable_ints()
 *
 *   Disable interrupts (if they are not already disabled).  And set the
 *   ints_disabled flag to TRUE, indicating that interrupts are no longer
 *   enabled.  Return the flag which is used to control wether or not interrupts
 *   should be re-enabled.
 *
 ******************************************************************************/

WORD disable_ints(void)
{
    if (!ints_disabled)
    {
        _disable();
        ints_disabled = TRUE;
        return(ints_disabled);
    }

    else
        return(FALSE);

}   /* End of disable_ints() */


/*******************************************************************************
 *
 *   void enable_ints()
 *
 *   Enables interrupts and marks the ints_disabled flag accordingly.
 *
 ******************************************************************************/

void enable_ints(void)
{
    _enable();
    ints_disabled = FALSE;

}   /* End of enable_ints() */
```

End of gen_apis.h, gen_apis.c

gen_defs.h

```
/*****************************************************************************
 *
 *    GEN_DEFS.H
 *
 *    Include file for general definitions used by most .h and .c files..
 *
 *    Petite Amateur Navy Satellite (PANSAT).
 *    Embedded ROM software.
 *    Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *    Jim A. Horning (Jah)
 *
 *    Revision History:
 *    =================
 *
 *    Date            Who     What
 *    ---------------+-------+-------------------------------------------
 *    8 Sept 1995     Jah     Creation
 *
 *****************************************************************************/


typedef  unsigned char       BYTE;
typedef  unsigned int     WORD;
typedef unsigned long int DWORD;


#define FALSE          0
#define TRUE           1
#define ERROR          0
#define NO_ERROR       1
#define OFF            0
#define ON             1
#define  RESET            0
#define  SET              1

#define  NULL          '\0'


/* Character definitions for ASCII */
#define NULL_CHAR       (char) 0x00
#define BELL            (char) 0x07
#define BACK_SPACE      (char) 0x08
#define LF              (char) 0x0A
#define CR              (char) 0x0D
#define ESC             (char) 0x1B
#define CTRL_Q          (char) 0x11
#define CTRL_R          (char) 0x12
#define CTRL_S          (char) 0x13
#define  CTRL_V            (char) 0x16
#define CTRL_W          (char) 0x17
#define CTRL_X          (char) 0x18
#define CTRL_Y          (char) 0x19
#define CTRL_Z          (char) 0x1A
#define HOME            (char) 0x1E
#define NEW_LINE        (char) 0x1F


#define  MAX_UCHAR         (unsigned char)            255
#define  MAX_CHAR       (signed char)          127
#define  MAX_UINT       (unsigned int)       65535
#define  MAX_INT          (int)              32767
#define  MAX_ULONG        (unsigned long int)  4294967295
#define  MAX_LONG       (long int)          2147483647


/* DMA Registers */
#define  D0SRCH    0xFFC2
#define  D0SRCL    0xFFC0
#define  D0DSTH    0xFFC6
#define  D0DSTL    0xFFC4
#define  D0CON     0xFFCA
#define  D0TC 0xFFC8
#define  D1SRCH    0xFFD2
#define  D1SRCL    0xFFD0
#define  D1DSTH    0xFFD6
#define  D1DSTL    0xFFD4
#define  D1CON     0xFFDA
#define  D1TC 0xFFD8
```

168

```
/* Operating Frequencies */
#define  CRYSTAL_FREQUENCY 14.7456E+6
/* Peripheral Clock: half the frequency of the crystal frequency */
#define  PCLK ((double)(1.0/(CRYSTAL_FREQUENCY/2.0)))
```

End of gen_defs.h

int.h, int.c

```
/***************************************************************************
 *
 *   INT.H
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *
 ***************************************************************************/


#define  INT_EOI      0xFF22
#define  INT_REQ_REG 0xFF2E


/***************************************************************************
 *
 *   INT.C
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *
 ***************************************************************************/

#include     "gen_defs.h"

#define      INT_C
#include "int.h"
#undef       INT_C
```

End of int.h, int.c

modem.h, modem.c

```
/***********************************************************************
 *
 *   MODEM.H
 *
 *   MODEM
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   17 July 1996 Jah       Creation
 *
 ***********************************************************************/


typedef struct
{
    BYTE address;
    BYTE data;

} pa100_instr_struct;


#ifdef   MODEM
        #define  PA100_BASE       0x200
        #define  MODEM_CTRL_PORT  0x180

        #define  MODEM_SETUP      0x00 /* Encode=OFF,DDS=OFF,Spread=OFF,Reset=OFF */
        #define  MODEM_RESET      0x01 /* Encode=OFF,DDS=OFF,Spread=OFF,Reset=ON */
        #define  MODEM_CLEAR      0x0C /* Encode=ON,DDS=ON,Spread=OFF,Reset=OFF */
        #define  MODEM_SPREAD 0x0E /* Encode=ON,DDS=ON,Spread=ON,Reset=OFF */

        void modem_clear(void);
        void modem_off(void);
        void modem_on(void);
        void modem_spread(void);

        void pa100_read_regs(void);
        void pa100_write_table(pa100_instr_struct table[]);


#endif


#ifndef  MODEM

        extern void  modem_clear(void);
        extern void  modem_off(void);
        extern void  modem_on(void);
        extern void  modem_spread(void);

        extern void  pa100_read_regs(void);
        extern void  pa100_write_table(pa100_instr_struct table[]);

#endif
```

171

```c
/****************************************************************************
 *
 *   MODEM.C
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   ==================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *   17 July 1996 Jah      Creation
 *
 ****************************************************************************/

#include         "gen_defs.h"


#define          MODEM
#include         "modem.h"
#undef           MODEM

#include        "pcb.h"


static pa100_instr_struct pa100_init[] =
{
     {0x01,   0xC6},   /* ALPHA_I FIR filter weights */
     {0x04,   0xC6},   /* ALPHA_Q */
     {0x02,   0x00},   /* IBI_LO  DC removal filter, I channel bias */
     {0x03,   0xE2},   /* IBI_HI */
     {0x05,   0x00},   /* QBI_LO  DC removal filter, Q channel bias */
     {0x06,   0xE2},   /* QBI_HI */
     /* AGC */
     {0x07,   0x22},   /* AGC_L   AGC reference Level */
     {0x08,   0x38},   /* AGC_G   AGC proportional & saturated gains */
     {0x09,   0x00},   /* AGC_I   AGC initial value */
     {0x0A,   0xF5},   /* GP1_CTL Init. AGC, apply DC removal filter initial bias */
     {0x0A,   0xF0},   /* GP1_CTL Let AGC free run, remove DC filter initial bias */
     /* PN Generators */
     {0x19,   0x00},   /* I_PNT_LO I PN generator taps */
     {0x1A,   0x82},   /* I_PNT_HI */
     {0x17,   0x00},   /* I_PNI_LO I PN generator initial register setting */
     {0x18,   0xBE},   /* I_PNI_HI */
     {0x1D,   0x00},   /* Q_PNT_LO Q PN generator taps */
     {0x1E,   0x82},   /* Q_PNT_HI */
     {0x1B,   0x00},   /* Q_PNI_LO Q PN generator initial register setting */
     {0x1C,   0xBE},   /* Q_PNI_HI */
     {0x1F,   0x77},   /* PN_CNTL0 */
     {0x20,   0x05},   /* PN_CNTL1 */
     /* PN Detector */
     {0x22,   0x00},   /* PNSEL PN detection code select (I or Q code) */
     /* General Controls */
     {0x28,   0x39},   /* CNTL_B2 Enable time/level/phase strobes, set polarities */
     {0x29,   0x01},   /* CNTL_B3 unfreeze, enable output data clock */
     {0x2A,   0x00},   /* EXT_IN  External data input controls */
     {0x2B,   0x00},   /* GP_3    Clear i/o muxing */
     {0x2C,   0xEA},   /* CNTL_AS Select symbol strobes used by accumulators */
     /* Phase Loop */
     {0x39,   0x00},   /* PH_FREQ_0 Phase accumulator initial value */
     {0x3A,   0x00},   /* PH_FREQ_1 */
     {0x3B,   0x00},   /* PH_FREQ_2 */
     {0x3C,   0x00},   /* PH_FREQ_3 */
     /* Time Loop */
     {0x14,   0xFA},   /* TIM_CTL2  Set subchip and chip counter sync. source */
     {0x33,   0x09},   /* TM_WIDTH  Freq Synth Ctrl Word width = 32-TM_WIDTH bits */
     {0x0C,   0x00},   /* SC_LO     Samples per subchip = 0 */
     {0x0D,   0x00},   /* SC_HI     Clear mode for initialization */
     {0x0E,   0x00},   /* CHIP      Subchips per chip = 0, clear mode for init */
     {0x0F,   0x00},   /* I_SYM_HI  I channel: chips per symbol = 0 */
     {0x10,   0x00},   /* I_SYM_LO  Clear mode for initialization */
     {0x11,   0x00},   /* Q_SYM_HI  Q channel: chips per symbol = 0 */
     {0x12,   0x00},   /* Q_SYM_LO */
     {0x15,   0x02},   /* TIM_CTL3  Toggle SYS_INIT, set I strobe via I, Q via Q */
     {0x15,   0x42},   /* TIM_CTL3  Set I & Q PN to (2^N)-1 or 2^N */
     {0x15,   0x02},   /* TIM_CTL3  Set I strobe from I channel, Q via Q channel */
     {0x2D,   0x00},   /* TM_FREQ_0 Rs, sample clock = 10 MHz for initialization */
     {0x2E,   0x00},   /* TM_FREQ_1 */
     {0x2F,   0x00},   /* TM_FREQ_2 */
     {0x30,   0x80},   /* TM_FREQ_3 */
     {0x31,   0x00},   /* TM_GAIN_1 Open Timing Loop */
     {0x32,   0x80},   /* TM_GAIN_2 Initialize Timing Loop */
```

172

```c
        {0x32,   0x00},   /* TM_GAIN_2 Stop initializing loop */

        {0xFF,   0xFF}    /* End */
};




static pa100_instr_struct pa100_clear[] =
{
        {0x0B,   0x00},   /* PA_SC      PREACCUMULATOR SCALING CONSTANT */
        /* PN GENERATORS */
        {0x21,   0x20},   /* CC         FREQ. DISC. ON,SET INT PN, I&Q PN ON/OFF */
        {0x26,   0x24},   /* CNTL_B0    QUADRAPHASE DATA, SEL LEVEL $ TIME CHANNEL */
        {0x27,   0x3B},   /* CNTL_B1    ENABLE DATA REMOVAL, SELECT PHASE CHANNEL */
        /* TIME ERROR DETECTOR PROCESSOR */
        {0x23,   0x55},   /* I_Q_TIME   SET SCALE FACTORS FOR TIMING ACCUMULATOR */
        /* PHASE LEVEL PROCESSOR */
        {0x24,   0x55},   /* I_Q_LEVEL  SET SCALE FACTORS FOR LEVEL ACCUMULATOR */
        {0x25,   0x55},   /* I_Q_PHASE  SET SCALE FACTORS FOR PHASE ACCUMULATOR */
        /* TIME LOOP */
        {0x0C,   0x3F},   /* SC_LO      Samples per subchip = 0 */
        {0x0D,   0x00},   /* SC_HI      clear mode for initialization */
        {0x0E,   0x00},   /* CHIP       Subchips per chip = 0, clear mode for init */
        {0x0F,   0x00},   /* I_SYM_HI   I channel: chips per symbol = 0 */
        {0x10,   0x01},   /* I_SYM_LO   clear mode for initialization */
        {0x11,   0x00},   /* Q_SYM_HI   Q channel: chips per symbol = 0 */
        {0x12,   0x01},   /* Q_SYM_LO */
        {0x13,   0x00},   /* TIM_CTL1   NO CODE SLIPPING, CLEAR MODE */
        {0x16,   0x00},   /* TIM_CTL4   CLEAR MODE, DISABLE */
        {0x13,   0x00},   /* TIM_CTL1   NO CODE SLIPPING */
        {0x2D,   0x00},   /* TM_FREQ_0  Rs, sample clock = START AT 10 MHz */
        {0x2E,   0x00},   /* TM_FREQ_1 */
        {0x2F,   0x00},   /* TM_FREQ_2 */
        {0x30,   0x80},   /* TM_FREQ_3 */
        {0x31,   0x58},   /* TM_GAIN_1  Open Loop, ARM TO CLOSE ON PN DET, SET K1 */
        {0x32,   0xD1},   /* TM_GAIN_2  LOAD FILTER WITH INITIAL VALUE, SET GAIN K2 */
        /* PN DETECTOR */
        {0x35,   0x00},   /* PNCD_BIAS  PN DETECTOR BIAS LEVEL */
        {0x36,   0x00},   /* PNCD_INITLO PN DETECTOR ACCUMULATOR */
        {0x37,   0x00},   /* PNCD_INITHI */
        {0x38,   0xFF},   /* PNCD_TIM   PN DETECTOR CORRELATION TIMER */
        /* PHASE LOOP */
        {0x3D,   0x7F},   /* PH_GAIN_1  CLOSE THE LOOP, MAKE LOOP FIRST ORDER */
        {0x3E,   0xD4},   /* PH_GAIN_2  LOAD ACCUM WITH INITIAL VALUE, SET GAIN K2 */
        /* TIME LOOP */
        {0x15,   0x42},   /* SYS_INIT */
        {0x15,   0x02},   /* SYS_INIT */
        /* PN DETECTOR */
        {0x34,   0x00},   /* PNCD_CTL   PN DETECTOR ACQ/TRACK CONTROLLER */
        {0x34,   0x04},   /* PNCD_CTL   RESTART TRACK SEQUENCE */
        {0x34,   0x00},   /* PNCD_CTL */
        /* TIME LOOP */
        {0x32,   0x51},   /* TM_GAIN_2  Stop LOADING TIME LOOP FILTER WITH INIT VALUE */
        /* FREQUENCY PULLIN/TRACK SETUP TABLE */
        /* PN GENERATORS */
        {0x21,   0x20},   /* CC         FREQ. DISC. ON,SET INT PN, I&Q PN ON/OFF */
        /* PHASE LEVEL PROCESSOR */
        {0x26,   0x28},   /* CNTL_B0    QUADRAPHASE DATA, SELECT LEVEL $ TIME CHANNEL */
        {0x27,   0x3B},   /* CNTL_B1    ENABLE DATA REMOVAL, SELECT PHASE CHANNEL */
        {0x3D,   0x7F},   /* PH_GAIN_1  CLOSE THE LOOP, MAKE LOOP FIRST ORDER */
        {0x3E,   0x54},   /* PH_GAIN_2  LOAD ACCUM WITH INITIAL VALUE, SET GAIN K2 */
        /* TIME ERROR DETECTOR PROCESSOR */
        {0x23,   0x55},   /* I_Q_TIME   SET SCALE FACTORS FOR TIMING ACCUMULATOR */
        /* PHASE LEVEL PROCESSOR */
        {0x24,   0x55},   /* I_Q_LEVEL  SET SCALE FACTORS FOR LEVEL ACCUMULATOR */
        {0x25,   0x55},   /* I_Q_PHASE  SET SCALE FACTORS FOR PHASE ACCUMULATOR */
        /* COHERENT TRACK SETUP TABLE */
        {0x3D,   0x54},   /* PH_GAIN_1  CLOSE THE LOOP, MAKE LOOP FIRST ORDER */
        {0x3E,   0x4E},   /* PH_GAIN_2  LOAD ACCUM WITH INITIAL VALUE, SET GAIN K2 */
        /* PHASE LEVEL PROCESSOR */
        {0x26,   0x00},   /* CNTL_B0    QUADRAPHASE DATA, SELECT LEVEL $ TIME CHANNEL */
        {0x27,   0x3D},   /* CNTL_B1    ENABLE DATA REMOVAL, SELECT PHASE CHANNEL */
        /* PN GENERATORS */
        {0x21,   0x00},   /* CC         FREQ. DISC. ON,SET INT PN, I&Q PN ON/OFF */
        /* TIME ERROR DETECTOR PROCESSOR */
        {0x23,   0x55},   /* I_Q_TIME   SET SCALE FACTORS FOR TIMING ACCUMULATOR */
        /* PHASE LEVEL PROCESSOR */
        {0x24,   0x55},   /* I_Q_LEVEL  SET SCALE FACTORS FOR LEVEL ACCUMULATOR */
        {0x25,   0x55},   /* I_Q_PHASE  SET SCALE FACTORS FOR PHASE ACCUMULATOR */

        {0xFF,   0xFF}    /* End */
};
```

173

```c
/*****************************************************************************
 *
 *   modem_clear()
 *
 *****************************************************************************/

void modem_clear(void)
{
    modem_on();

    /* outp(MODEM_CTRL_PORT, MODEM_SETUP); */  /* done by modem_on() */

    pa100_write_table(pa100_init);

    outp(MODEM_CTRL_PORT, MODEM_CLEAR);

    pa100_write_table(pa100_clear);

}   /* End of modem_clear() */


/*****************************************************************************
 *
 *   modem_off()
 *
 *****************************************************************************/

void modem_off(void)
{
    pcb_portc(0, ON);    /* Turn this control bit ON to turn OFF modem */

}   /* End of modem_off() */


/*****************************************************************************
 *
 *   modem_on()
 *
 *****************************************************************************/

void modem_on(void)
{
    WORD x;


    pcb_portc(0, OFF);        /* Turn this  control bit OFF to turn ON modem */

    for (x = 0; x < 0xFFFF; x++)
        ;

    outp(MODEM_CTRL_PORT, MODEM_RESET);

    for (x = 0; x < 0xFFFF; x++)
        ;

    outp(MODEM_CTRL_PORT, MODEM_SETUP);

}   /* End of modem_on() */


/*****************************************************************************
 *
 *   modem_spread()
 *
 *****************************************************************************/

void modem_spread(void)
{
    /* normal spread, fixed encode, on DDS on*/
    outp(MODEM_CTRL_PORT, MODEM_SETUP);

    pa100_write_table(pa100_init);

    outp(MODEM_CTRL_PORT, MODEM_SPREAD);

}   /* End of modem_spread() */


/*****************************************************************************
 *
 *   pa100_read_regs()
 *
 *****************************************************************************/
```

174

```c
void    pa100_read_regs(void)
{
    unsigned int    addr, data, x;
    unsigned int data_tab[0x20];
    unsigned int b;
    unsigned long    a;
    double           f;


    /* Freeze */
    outp(PA100_BASE + (0x29)<<1, 0x81);
    for (x = 1; x <= 0x14; x++)
        data_tab[x] = inp(PA100_BASE + (x<<1));
    /* Unfreeze */
    outp(PA100_BASE + (0x29)<<2, 0x01);


    for (x = 1; x <= 0x14; x++)
    {
        data = data_tab[x];

        switch(x)
        {
            case 1:
                dprint("01: AGC Status                = %02Xh\n", data);
                break;

            case 2:
                b = data;
                break;

            case 3:
                b = (b<<8) + data;
                dprint("02, 03: I Prefilter           = %04Xh\n", b);
                break;

            case 4:
                b = data;
                break;

            case 5:
                b = (b<<8) + data;
                dprint("04, 05: Q Prefilter           = %04Xh\n", b);
                break;

            case 6:
                a = data;
                break;

            case 7:
                a = (((unsigned long)data) << 8) + a;
                break;

            case 8:
                a = (((unsigned long)data) << 16) + a;
                break;

            case 9:
                a = (((unsigned long)data) << 24) + a;
                f = ((double)a)/((double)214.748365E6);
                dprint("06 - 09 Time Frequency Command   = %lXh\n", a);
                dprint("                                 = %lf MHz\n", f);
                break;

            case 0xA:
                a = data;
                break;

            case 0xB:
                a = (((unsigned long)data) << 8) + a;
                break;

            case 0xC:
                a = (((unsigned long)data) << 16) + a;
                break;

            case 0xD:
                a = (((unsigned long)data) << 24) + a;
                f = ((double)a)/((double)214.748365E6);
                dprint("0A - 0D Phase Frequency Command       = %lXh\n", a);
                dprint("                                      = %lf MHz\n", f);
                break;

            case 0xE:
```

175

```
                        break;

            case 0xF:
                break;

            case 0x10:
                b = data;
                break;

            case 0x11:
                b = (data << 8) + b;
                dprint("10, 11: PN Correlation Detector  = %04Xh, %u\n", b, b);
                break;

            case 0x12:
                b = data;
                break;

            case 0x13:
                b = (data << 8) + b;
                dprint("12, 13: PN Correlation Slip      = %04Xh, %u\n", b, b);
                break;

            case 0x14:
                dprint("14: PN Generator Status          = %02Xh\n", data);
                if (data & 0x01)
                    dprint("     IPN_EP_TOG = 1\n");
                else
                    dprint("     IPN_EP_TOG = 0\n");

                if (data & 0x02)
                    dprint("     QPN_EP_TOG = 1\n");
                else
                    dprint("     QPN_EP_TOG = 0\n");
                break;
        }

    }

}   /* End of pa100_read_regs() */


/***************************************************************************
 *
 *   pa100_write_table()
 *
 ***************************************************************************/

void pa100_write_table(pa100_instr_struct table[])
{
    int     x;

    for (x = 0; table[x].address != (BYTE)0xFF; x++)
    {
        if (table[x].address == 0)
            outp(MODEM_CTRL_PORT, (WORD)table[x].data);
        else
            outp(PA100_BASE + (WORD)(table[x].address<<1), (WORD)table[x].data);
    }

}   /* End of pa100_write_table() */
```

End of modem.h, modem.c

msu.h, msu.c

```
/****************************************************************************
 *
 *   MSU.H
 *
 *   Include file for Mass Storage Units (MSU) software interface.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+-----------------------------------------------
 *   5 Sept 1996     Jah     Creation
 *
 ****************************************************************************/


#ifdef   MSU
     #define  NUM_FLASH_DEVICES          4

     #define  FLASH_SECTOR_SIZE          ((DWORD)0x4000L)          /* 16 kbytes */
     #define  FLASH_SECTOR_END_ADDRESS   ((DWORD)(FLASH_SECTOR_SIZE - 1))
     #define  FLASH_SECTORS_PER_DEVICE   8
     #define  FLASH_SECTOR_MAX_PER_DEVICE    (FLASH_DEVICE_SECTORS - 1)
     #define  FLASH_DEVICE_SIZE          (FLASH_SECTORS_PER_DEVICE * FLASH_SECTOR_SIZE)

     #define  FLASH_SIZE                 (NUM_FLASH_DEVICES * FLASH_DEVICE_SIZE)
     #define  FLASH_END_ADDRESS          (FLASH_SIZE - 1)
     #define  FLASH_SECTORS              (NUM_FLASH_DEVICES * FLASH_SECTORS_PER_DEVICE)
     #define  FLASH_SECTOR_MAX           (FLASH_SECTORS - 1)

     #define  ERASE_TIME_LIMIT           ((DWORD)0x0003FFFFL)
     #define  WRITE_TIME_LIMIT           0x0FFF

     /* Masks to signify the search method that found the first empty record */
     #define  NO_TLM_WRAP                0x0
     #define  TLM_WRAP                0x8000
     #define  NO_TLM_FIND                0xFFFF
     #define  NO_REC_NUM                 0xFFFF

     void msu_init(int device);
     void    msu_on(int device);
     void    msu_off(int device);

     int     msu_flash_erase(int device);
     int     msu_flash_erase_sector(int device, int sector);
     BYTE msu_flash_read1(int device, DWORD addr);
     void msu_flash_read(int device, DWORD addr, BYTE *buf, int count);
     int     msu_flash_write1(int device, DWORD addr, BYTE data);
     int     msu_flash_write(int device, DWORD addr, BYTE *data, int count);
     void msu_set_faddr(int device, DWORD addr);

     int     msu_calc_first_rec(int rec_num);
     int     msu_check_flash_tlm(void);
     WORD msu_flash_search(int device);




     int     msu_get_tlm(tlm_record_struct *r_tlm, int rec_num);
     void msu_save_tlm(tlm_record_struct *r_tlm);

     BYTE msu_sram_read1(int device, DWORD addr);
     void msu_sram_read(int device, DWORD addr, BYTE *buf, int count);
     int     msu_sram_write1(int device, DWORD addr, BYTE data);
     int     msu_sram_write(int device, DWORD addr, BYTE *data, int count);
     void msu_set_saddr(int device, DWORD addr);

     void msu_ftest(int device);
     void msu_stest(int device);


#endif

/* prototypes for modules other than msu.c */
#ifndef MSU
     extern void      msu_init(int device);
```

```
        extern void       msu_on(int device);
        extern void       msu_off(int device);

        extern int        msu_flash_erase(int device);
        extern int        msu_flash_erase_sector(int device, int sector);
        extern BYTE       msu_flash_read1(int device, DWORD addr);
        extern void       msu_flash_read(int device, DWORD addr, BYTE *buf, int count);
        extern int        msu_flash_write1(int device, DWORD addr, BYTE data);
        extern int        msu_flash_write(int device, DWORD addr, BYTE *data, int count);
        extern void       msu_set_faddr(int device, DWORD addr);

        extern int        msu_check_flash_tlm(void);
        extern int        msu_get_tlm(tlm_record_struct *r_tlm, int rec_num);
        extern void       msu_save_tlm(tlm_record_struct *r_tlm);

        extern BYTE       msu_sram_read1(int device, DWORD addr);
        extern void       msu_sram_read(int device, DWORD addr, BYTE *buf, int count);
        extern int        msu_sram_write1(int device, DWORD addr, BYTE data);
        extern int        msu_sram_write(int device, DWORD addr, BYTE *data, int count);
        extern void       msu_set_saddr(int device, DWORD addr);

        extern void       msu_ftest(int device);
        extern void       msu_stest(int device);

#endif


/****************************************************************************
 *
 *   MSU.C
 *
 *   Interface for the Mass Storage Units (MSU).
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   The Mass Storage Units have 4 Mbytes of SRAM and 1/2 Mbyte of Flash.
 *   The Flash devices are the Am29F010.
 *
 *
 *   Revision History:
 *   ==================
 *   Date           Who     What
 *   ---------------+-------+----------------------------------------------
 *   5 Sept 1996    Jah     Creation
 *   22 April 1997  Jah     Support for record keeping.
 *
 ****************************************************************************/

#include        "gen_defs.h"

#include        "bcm.h"
#include        "tlm.h"

#define         MSU
#include        "msu.h"
#undef          MSU

#include        "dcs.h"
#include        "eps.h"
#include        "pcb.h"



/* Flash storage telemtry record pointers */
WORD msu_tlm_rec_num = 0;                /* current location to record to */
WORD msu_tlm_first_rec_num = 0;            /* location of oldest record */
WORD msu_tlm_last_rec_num = NO_REC_NUM; /* location of newest record */

#define  LAST_TLM_REC_NUM  ((FLASH_DEVICE_SIZE/sizeof(tlm_record_struct)) - 1)

/****************************************************************************
 *
 *   void    msu_init(int device)
 *
 *   Initializes a MSU.
 *
 ****************************************************************************/

void    msu_init(int device)
{
    pcb_write(device, 3, 0x80);
```

```
        pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

}   /* End of msu_init() */


/*****************************************************************************
 *
 *   void    msu_on(int device)
 *
 *   Turns ON and initializes a MSU.
 *
 *****************************************************************************/

void    msu_on(int device)
{
    WORD i;


    if (device == MSA0)
        eps_set_power(PWR_MSA, ON);
    else
        eps_set_power(PWR_MSB, ON);

    for (i = 0; i < 0x1FFF; i++)   /* pause for power ON */
        ;
    msu_init(device);

}   /* End of msu_on() */


/*****************************************************************************
 *
 *   void    msu_off(int device)
 *
 *   Turns OFF a MSU.
 *
 *****************************************************************************/

void    msu_off(int device)
{
    if (device == MSA0)
        eps_set_power(PWR_MSA, OFF);
    else
        eps_set_power(PWR_MSB, OFF);

}   /* End of msu_off() */


/*****************************************************************************
 *
 *   WORD msu_flash_codes(int device)
 *
 *   Examines the four Flash devices for a MSU to see if the manufacturer
 *   code (0x01 = AMD) and the device type (0x20 = 29F010) are readable from
 *   each.
 *
 *****************************************************************************/

WORD msu_flash_codes(int device)
{
    register int i;
    BYTE         data;
    register WORD    flag = 0;


    for (i = 0; i < 4; i++)
    {
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, ((BYTE)0xAA));
        msu_set_faddr(device, ((i*0x00020000L) + 0x00002AAAL));
        pcb_write(device+1, 0, ((BYTE)0x55));
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, ((BYTE)0x90));

        msu_set_faddr(device, (i*0x00020000L));
        data = pcb_read(device+1, 0);

        if (data == 0x01)
        {
            msu_set_faddr(device, ((i*0x00020000L) + 1L));
            data = pcb_read(device+1, 0);

            if (data == 0x20)
                flag |= (1<<i);
```

179

```
                else
                {
                    flag |= 0x80;
                    break;
                }
            }

            else
            {
                flag |= 0x80;
                break;
            }

            /* perform read/reset */
            msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
            pcb_write(device+1, 0, ((BYTE)0xAA));
            msu_set_faddr(device, ((i*0x00020000L) + 0x00002AAAL));
            pcb_write(device+1, 0, ((BYTE)0x55));
            msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
            pcb_write(device+1, 0, ((BYTE)0xF0));

        }

        pcb_write(device, 2, 0x48);         /* Point to landing zone for low power */

        return(flag);

}   /* End of msu_flash_codes() */


/****************************************************************************
 *
 *   int      msu_flash_erase(int device)
 *
 *   Erases all Flash devices within a MSU.
 *
 ****************************************************************************/

int      msu_flash_erase(int device)
{
    register BYTE      fdata;
    register int i;
    int                pass = TRUE;
    DWORD              x;


    for (i = 0; (i < 4) && pass; i++)
    {
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, (BYTE)0xAA);
        msu_set_faddr(device, ((i*0x00020000L) + 0x00002AAAL));
        pcb_write(device+1, 0, (BYTE)0x55);
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, (BYTE)0x80);
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, (BYTE)0xAA);
        msu_set_faddr(device, ((i*0x00020000L) + 0x00002AAAL));
        pcb_write(device+1, 0, (BYTE)0x55);
        msu_set_faddr(device, ((i*0x00020000L) + 0x00005555L));
        pcb_write(device+1, 0, (BYTE)0x10);


        fdata = pcb_read(device+1, 0);
        x = 0;
        while (((fdata & 0x80) != 0x80) && (x < ERASE_TIME_LIMIT))
        {
            if (fdata & 0x20)
            {
                fdata = pcb_read(device+1, 0);
                if ((fdata & 0x80) == 0x80)
                    break;

                else
                {
                    pass = FALSE;
                    break;
                }
            }

            fdata = pcb_read(device+1, 0);
            x++;

        }   /* End of WHILE */
```

180

```
            if (x == ERASE_TIME_LIMIT)
                pass = FALSE;

        }   /* End of FOR */

        pcb_write(device, 2, 0x48);         /* Point to landing zone for low power */

        if (x == ERASE_TIME_LIMIT)
            return(0xC0 | i);

        else if (!pass)
            return(0x80 | i);    /* 0x80 = Error flag, i = device which failed */

        else
            return(pass);

}   /* End of msu_flash_erase() */


/***************************************************************************
 *
 *   int msu_flash_write1(int device, DWORD addr, BYTE data)
 *
 *   Write one data byte to a flash address.
 *
 ***************************************************************************/

int msu_flash_write1(int device, DWORD addr, BYTE data)
{
    register BYTE    fdata;
    int              pass = TRUE;
    register WORD    x;


    msu_set_faddr(device, (addr&0x00070000L) + 0x5555);
    pcb_write(device+1, 0, (BYTE)0xAA);
    msu_set_faddr(device, (addr&0x00070000L) + 0x2AAA);
    pcb_write(device+1, 0, (BYTE)0x55);
    msu_set_faddr(device, (addr&0x00070000L) + 0x5555);
    pcb_write(device+1, 0, (BYTE)0xA0);

    msu_set_faddr(device, addr);
    pcb_write(device+1, 0, data);

    fdata = pcb_read(device+1, 0);
    x = 0;
    while (((fdata & 0x80) != (data & 0x80)) && (x < WRITE_TIME_LIMIT))
    {
        if (fdata & 0x20)
        {
            fdata = pcb_read(device+1, 0);
            if ((fdata & 0x80) == (data & 0x80))
                break;

            else
            {
                pass = FALSE;
                break;
            }
        }

        fdata = pcb_read(device+1, 0);
        x++;
    }

    pcb_write(device, 2, 0x48);         /* Point to landing zone for low power */

    if (x == WRITE_TIME_LIMIT)
        pass = FALSE;

    return(pass);

}   /* End of msu_flash_write1() */


/***************************************************************************
 *
 *   int msu_flash_write(int device, DWORD addr, BYTE *data, int count)
 *
 *   Write data to a flash address(es).
 *
 ***************************************************************************/

int msu_flash_write(int device, DWORD addr, BYTE *data, int count)
```

181

```
{
    register BYTE    fdata;
    int              pass = TRUE;
    register WORD    x;


    x = 0;
    while ((count--) && (x < WRITE_TIME_LIMIT))
    {
        msu_set_faddr(device, (addr&0x00070000L) + 0x5555);
        pcb_write(device+1, 0, (BYTE)0xAA);
        msu_set_faddr(device, (addr&0x00070000L) + 0x2AAA);
        pcb_write(device+1, 0, (BYTE)0x55);
        msu_set_faddr(device, (addr&0x00070000L) + 0x5555);
        pcb_write(device+1, 0, (BYTE)0xA0);

        msu_set_faddr(device, addr);
        pcb_write(device+1, 0, *data);

        fdata = pcb_read(device+1, 0);
        x = 0;
        while (((fdata & 0x80) != (*data & 0x80)) && (x < WRITE_TIME_LIMIT))
        {
            if (fdata & 0x20)
            {
                fdata = pcb_read(device+1, 0);
                if ((fdata & 0x80) == (*data & 0x80))
                    break;

                else
                {
                    pass = FALSE;
                    break;
                }
            }

            fdata = pcb_read(device+1, 0);
            x++;
        }

        addr++;
        data++;
    }

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

    if (x == WRITE_TIME_LIMIT)
        pass = FALSE;

    return(pass);

}   /* End of msu_flash_write() */


/****************************************************************************
 *
 *   BYTE msu_flash_read1(int device, DWORD addr)
 *
 *   Read one data byte from a flash address.
 *
 ****************************************************************************/

BYTE msu_flash_read1(int device, DWORD addr)
{
    register BYTE    data;


    msu_set_faddr(device, addr);
    data = pcb_read(device+1, 0);

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

    return(data);

}   /* End of msu_flash_read1() */


/****************************************************************************
 *
 *   void msu_flash_read(int device, DWORD addr, BYTE *buf, int count)
 *
 *   Read data from a flash address.  This routine only increments the address
 *   on the MSU if it has changed; thereby reducing many PCB Writes.  The
 *   overhead to check for a,c,b address roll-over is nothing compared to the
```

182

```
 *   PCB Write.
 *
 *****************************************************************************/

voidmsu_flash_read(int device, DWORD addr, BYTE *buf, int count)
{
    register WORD    a, b;
    WORD            c;


    msu_set_faddr(device, addr);
    a = (WORD)((addr) & 0x000000FFL);
    b = (WORD)(((addr) & 0x0000FF00L)>>8);
    c = (WORD)((((addr) | 0x00400000L) & 0x00FF0000L)>>16);

    while(count--)
    {
        *buf = pcb_read(device+1, 0);
        buf++;

        /* Now, setup for the next address to write to */
        a++;
        pcb_write(device, 0, a%0x100);
        if (a > 0xFF)
        {
            a = 0;
            b++;
            pcb_write(device, 1, b%0x100);
            if (b > 0xFF)
            {
                b = 0;
                c++;
                pcb_write(device, 2, c);
            }
        }

    }

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

}  /* End of msu_flash_read() */


/*****************************************************************************
 *
 *   BYTEmsu_sram_read1(int device, DWORD addr)
 *
 *   Read data to a flash address.
 *
 *****************************************************************************/

BYTEmsu_sram_read1(int device, DWORD addr)
{
    register BYTE    data;


    msu_set_saddr(device, addr);
    data = pcb_read(device+1, 0);

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

    return(data);

}  /* End of msu_sram_read1() */


/*****************************************************************************
 *
 *   voidmsu_sram_read(int device, DWORD addr, BYTE *buf, int count)
 *
 *   Read data to a flash address.  This routine only increments the address
 *   on the MSU if it has changed; thereby reducing many PCB Writes.  The
 *   overhead to check for a,c,b address roll-over is nothing compared to the
 *   PCB Write.
 *
 *****************************************************************************/

voidmsu_sram_read(int device, DWORD addr, BYTE *buf, int count)
{
    register WORD    a, b;
    WORD            c;


    msu_set_saddr(device, addr);
```

```
        a = (WORD)((addr) & 0x000000FFL);
        b = (WORD)(((addr) & 0x0000FF00L)>>8);
        c = (WORD)(((addr) & 0x00FF0000L)>>16);

    while(count--)
    {
        *buf = pcb_read(device+1, 0);
        buf++;

        /* Now, setup the next address to write to */
        a++;
        pcb_write(device, 0, a%0x100);
        if (a > 0xFF)
        {
            a = 0;
            b++;
            pcb_write(device, 1, b%0x100);
            if (b > 0xFF)
            {
                b = 0;
                c++;
                pcb_write(device, 2, c);
            }
        }

    }

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

}   /* End of msu_sram_read() */


/***************************************************************************
 *
 *   int msu_sram_write1(int device, DWORD addr, BYTE data)
 *
 *   Write data to a sram address.
 *
 ***************************************************************************/

int msu_sram_write1(int device, DWORD addr, BYTE data)
{
    msu_set_saddr(device, addr);
    pcb_write(device+1, 0, data);

    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

}   /* End of msu_sram_write1() */


/***************************************************************************
 *
 *   int msu_sram_write(int device, DWORD addr, BYTE *data, int count)
 *
 *   Write data to a sram address.  This routine only increments the address
 *   on the MSU if it has changed; thereby reducing many PCB Writes.  The
 *   overhead to check for a,c,b address roll-over is nothing compared to the
 *   PCB Write.
 *
 ***************************************************************************/

int msu_sram_write(int device, DWORD addr, BYTE *data, int count)
{
    register WORD    a, b;
    WORD          c;


    msu_set_saddr(device, addr);
    a = (WORD)((addr) & 0x000000FFL);
    b = (WORD)(((addr) & 0x0000FF00L)>>8);
    c = (WORD)(((addr) & 0x00FF0000L)>>16);

    while(count--)
    {
        pcb_write(device+1, 0, *data);
        data++;

        /* Now, setup the next address to write to */
        a++;
        pcb_write(device, 0, a%0x100);
        if (a > 0xFF)
        {
            a = 0;
            b++;
```

```
                pcb_write(device, 1, b%0x100);
                if (b > 0xFF)
                {
                    b = 0;
                    c++;
                    pcb_write(device, 2, c);
                }
            }

        }

        pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */

}   /* End of msu_sram_write() */


/*******************************************************************************
 *
 *   void msu_ftest(int device)
 *
 *   Tests a MSU's Flash devices by writing a pattern to the Flash devices
 *   and then reading it back.
 *
 *******************************************************************************/

void msu_ftest(int device)
{
    DWORD         addr;
    int           i, j;
    BYTE block[256];
    BYTE fblock[256];
    DWORD     t;
    extern int    icount;


    for (i = 0; i < 255; i++)
        block[i] = (BYTE)i;
    block[255] = 0xFE;              /* don't use 0xFF which is erased value */


    t = icount;
    msu_flash_write(device, 0, (BYTE *)block, 256);
    t = icount - t;
    dprint("FLASH 256 byte block write time = %ld ticks.\n", t);
    t = icount;
    msu_flash_read(device, 0, (BYTE *)block, 256);
    t = icount - t;
    dprint("FLASH 256 byte block read time = %ld ticks.\n", t);
    return;


    dprint("Writing Flash data (# = 64K)\n");
    for (i = 0, addr = 0L; addr <= 0x0007FFFFL; addr += 256)
    {
        msu_flash_write(device, addr, (BYTE *)block, 256);

        if (++i == 256)
        {
            i = 0;
            dprint("#");
        }
    }

    dprint("\nReading back Flash data (# = 64K)\n");

    for (i = 0, addr = 0L; addr <= 0x0007FFFF; addr += 256)
    {
        msu_flash_read(device, addr, (BYTE *)fblock, 256);
        for (j = 0; j < 256; j++)
        {
            if (fblock[j] != block[j])
            {
                dprint("Flash read back error at %lX, %X should be %X\n",
                    (DWORD)(addr+j), fblock[j], block[j]);
                return;
            }
        }
        if (++i == 256)
        {
            i = 0;
            dprint("#");
        }
    }
```

185

```
        dprint("\nFinished: OK.\n");

}   /* End of msu_ftest() */


/*****************************************************************************
 *
 *  void msu_stest(int device)
 *
 *  Tests a MSU's SRAM devices by writing a pattern to the SRAM devices
 *  and then reading it back.
 *
 *****************************************************************************/

void msu_stest(int device)
{
    DWORD       addr;
    int         i, j;
    BYTE block[256];
    BYTE fblock[256];
    DWORD   t;
    #define  LAST_ADDR   0x0002FFFFL
    extern int   icount;


    for (i = 0; i < 256; i++)
        block[i] = (BYTE)i;


    t = icount;
    msu_sram_write(device, 0, (BYTE *)block, 256);
    t = icount - t;
    dprint("SRAM 256 byte block write time = %ld ticks.\n", t);
    t = icount;
    msu_sram_read(device, 0, (BYTE *)block, 256);
    t = icount - t;
    dprint("SRAM 256 byte block read time = %ld ticks.\n", t);
    return;


    dprint("Writing SRAM data (# = 64K)\n");
    for (i = 0, addr = 0L; addr <= LAST_ADDR; addr += 256)
    {
        msu_sram_write(device, addr, (BYTE *)block, 256);

        if (++i == 256)
        {
            i = 0;
            dprint("#");
        }
    }

    dprint("\nReading back SRAM data (# = 64K)\n");

    for (i = 0, addr = 0L; addr <= LAST_ADDR; addr += 256)
    {
        msu_sram_read(device, addr, (BYTE *)fblock, 256);
        for (j = 0; j < 256; j++)
        {
            if (fblock[j] != block[j])
            {
                dprint("SRAM read back error at %lX, %X should be %X\n",
                    (DWORD)(addr+j), fblock[j], block[j]);
                return;
            }
        }
        if (++i == 256)
        {
            i = 0;
            dprint("#");
        }
    }

    dprint("\nFinished: OK.\n");

}   /* End of msu_stest() */


/*****************************************************************************
 *
 *  void msu_set_saddr(int device, DWORD addr)
 *
 *  Set an address to the SRAM array on a particular device.
 *
```

186

```
**********************************************************************/

void    msu_set_saddr(int device, DWORD addr)
{
    pcb_write(device, 0, (WORD)((addr) & 0x000000FFL));
    pcb_write(device, 1, (WORD)(((addr) & 0x0000FF00L)>>8));
    pcb_write(device, 2, (WORD)(((addr) & 0x00FF0000L)>>16));

}   /* End of msu_set_saddr() */


/*********************************************************************
 *
 *   void msu_set_faddr(int device, DWORD addr)
 *
 *   Set an address to the Flash array on a particular device.
 *
 *********************************************************************/

void msu_set_faddr(int device, DWORD addr)
{
    pcb_write(device, 0, (WORD)((addr) & 0x000000FFL));
    pcb_write(device, 1, (WORD)(((addr) & 0x0000FF00L)>>8));
    pcb_write(device, 2, (WORD)((((addr) | 0x00400000L) & 0x00FF0000L)>>16));

}   /* End of msu_set_faddr() */


/*********************************************************************
 *
 *   void msu_save_tlm()
 *
 *   Save a telemetry structure to Mass Storage.
 *
 *********************************************************************/

void msu_save_tlm(tlm_record_struct *r_tlm)
{
    int     sector;
    int     next_sector;
    int     remaining;
    DWORD   msu_ptr;


    /* Calculate, and add CRC to end of the record */
    prepare_crc(r_tlm, sizeof(tlm_record_struct) -2);   /* -2 due to CRC */


    /* Is there enough room on the existing sector to write out this telemetry
     * record, and the next record ?
     */
    msu_ptr = (DWORD)(msu_tlm_rec_num * sizeof(tlm_record_struct));
    remaining = msu_ptr%FLASH_SECTOR_SIZE;
    if (remaining < 2*sizeof(tlm_record_struct))
    {
        /* There is not enough room for this and the next record.  Erase the
         * next highest sector (unless it is time to wrap around).
         */
        sector = msu_ptr/FLASH_SECTOR_SIZE;
        if (sector == FLASH_SECTOR_MAX)
            next_sector = 0;
        else
            next_sector = sector+1;

        /* erase the "next" sector */
        msu_flash_erase_sector(MSA, next_sector);
        msu_flash_erase_sector(MSB, next_sector);
    }


    /* Save the record to both MSA and MSB */
    msu_flash_write(MSA, msu_ptr, (BYTE *)r_tlm, sizeof(tlm_record_struct));
    msu_flash_write(MSB, msu_ptr, (BYTE *)r_tlm, sizeof(tlm_record_struct));


    /* Update counter for next time */
    if (next_sector == 0)
        msu_tlm_rec_num = 0;
    else
        msu_tlm_rec_num++;

}   /* End of msu_save_tlm() */
```

```
/*****************************************************************************
 *
 *   int msu_get_tlm()
 *
 *   Retrieve a telemetry structure from Mass Storage.
 *
 *****************************************************************************/

int msu_get_tlm(tlm_record_struct *r_tlm, int rec_num)
{
    int      flag = NO_ERROR;
    DWORD    msu_ptr;


    msu_ptr = (DWORD)(rec_num * sizeof(tlm_record_struct));

    msu_flash_read(MSA, msu_ptr, (BYTE*)r_tlm, sizeof(tlm_record_struct));
    if (check_crc(r_tlm, sizeof(tlm_record_struct)) != 0)
    {
        /* Try the other Mass Storage Flash */
        msu_flash_read(MSB, msu_ptr, (BYTE*)r_tlm, sizeof(tlm_record_struct));
        if (check_crc(r_tlm, sizeof(tlm_record_struct)) != 0)
            flag = ERROR;
    }

    return(flag);

}   /* End of msu_get_tlm() */


/*****************************************************************************
 *
 *   int msu_check_flash_tlm()
 *
 *   Check Flash for already stored telemetry records.  This is first called
 *   when the system is Reset to see if any prior state history has already
 *   been saved to the Flash.
 *
 *****************************************************************************/

int msu_check_flash_tlm(void)
{
    int      rec_num;
    DWORD    msu_ptr;
    int      tryb = FALSE;
    DWORD    etime;
    int      flag = NO_ERROR;
    int      wrap = FALSE;
    tlm_record_struct rtlm;
    tlm_record_struct *r_tlm;


    /* First, check using MSA. */
    /* Do a search looking for unused portions of the Flash. */
    if ((rec_num = msu_flash_search(MSA)) != NO_TLM_FIND)
    {
        wrap = (rec_num & TLM_WRAP) ? TRUE : FALSE;
        rec_num &= ~TLM_WRAP;

        /* MSA has an empty location. */
        msu_ptr = (DWORD)(rec_num * sizeof(tlm_record_struct));

        msu_flash_read(MSA, msu_ptr, (BYTE*)r_tlm, sizeof(tlm_record_struct));
        if (check_crc(r_tlm, sizeof(tlm_record_struct)) != 0)
            tryb = TRUE;
    }
    else
        tryb = TRUE;      /* attempt the same search with MSB. */


    if (tryb)
    {
        if ((rec_num = msu_flash_search(MSB)) == NO_TLM_FIND)
        {
            /* ERROR.  Erase both MSA and MSB.  And assume NO recorded data. */
            msu_flash_erase(MSA);
            msu_flash_erase(MSB);
            rec_num = 0;
            flag = ERROR;
        }

        /* MSB has an empty location. */
        wrap = (rec_num & TLM_WRAP) ? TRUE : FALSE;
        rec_num &= ~TLM_WRAP;
```

188

```c
            msu_ptr = (DWORD)(rec_num * sizeof(tlm_record_struct));

            msu_flash_read(MSB, msu_ptr, (BYTE*)r_tlm, sizeof(tlm_record_struct));
            if (check_crc(r_tlm, sizeof(tlm_record_struct)) != 0)
            {
                /* ERROR: Erase both MSA and MSB.  And assume NO recorded data. */
                msu_flash_erase(MSA);
                msu_flash_erase(MSB);
                rec_num = 0;
                flag = ERROR;
            }
            else
            {
                /* MSA had problems, but not MSB.  So, erase MSA. */
                msu_flash_erase(MSA);
            }
    }


    /* Now, process the last recorded record by using it as the most recently saved
     * history regarding the state of the spacecraft.
     */


    /* If the record number returned from the search is not zero, then
     * assume that there is a prior record and this is not the first time
     * recording.
     * If the record number returned from the search is zero, then this could be
     * the first time recording (or at least first timer recording since the
     * Flash was erased).  And thus, there is no history of data to examine.
     */

    if (flag == ERROR)
    {
        /* No state history, both MSA and MSB have just been erased.
         * Begin as if ejection has just occurred.
         */
        msu_tlm_rec_num = 0;
        msu_tlm_last_rec_num = NO_REC_NUM;
        msu_tlm_first_rec_num = 0;
    }

    else if (wrap == FALSE)
    {
        if (rec_num == 0)
        {
            /* Empty Flash */
            msu_tlm_rec_num = 0;
            msu_tlm_last_rec_num = NO_REC_NUM;
            msu_tlm_first_rec_num = 0;
        }

        else
        {
            /* Flash has data, but no wrap around is in effect */
            msu_tlm_rec_num = rec_num;
            msu_tlm_last_rec_num = rec_num - 1;
            msu_tlm_first_rec_num = 0;
            msu_get_tlm(&tlm_record, msu_tlm_last_rec_num);
        }
    }

    else if (wrap == TRUE)
    {
        /* Flash has data, and wrap around is in effect */
        msu_tlm_rec_num = rec_num;
        if (rec_num == 0)
            msu_tlm_last_rec_num = 0;
        else
            msu_tlm_last_rec_num = rec_num - 1;
        /* calculate msu_tlm_first_rec_num */
        msu_tlm_first_rec_num = msu_calc_first_rec(rec_num);
        msu_get_tlm(&tlm_record, msu_tlm_last_rec_num);
    }


}  /* End of msu_check_flash_tlm() */


/*****************************************************************************
 *
 *  msu_flash_search()
 *
 *  Check Flash via a "top" binary search.  That is, use an increasing memory
 *  binary search to see if any "empty" tlm records exists (not recorded yet).
```

```
*    When performing the "divide and conquer", always take the upper portion of
*    memory.
*
*    If no "empty" record is found, perform a search starting at the bottom and
*    look for the first "empty" record.  This is not a fast search (its linear),
*    but there is no way around it since one has no idea where the first hole
*    could be and there is no ordering.
*
*    The "empty" record number is returned.
*
*****************************************************************************/

WORDmsu_flash_search(int device)
{
    WORDbottom, top, mid, r, sector;
    DWORD    etime;
    DWORD    msu_ptr;


    /* Binary Search */
    /* These are the end points of the tlm record storage.  Attempt to find
     * an "empty" storage record.  Assuming no lower "bottom" holes exist.
     */
    bottom = 0;
    top = FLASH_SIZE/sizeof(tlm_record_struct);
    while ((top - bottom) > 1)
    {
        mid = (bottom + top)/2;
        msu_ptr = mid*sizeof(tlm_record_struct);
        msu_flash_read(device, msu_ptr, &etime, sizeof(DWORD));
        if (etime != 0xFFFFFFFFL) /* in use, look higher */
            bottom = mid;
        else
            top = mid;
    }

    msu_ptr = bottom*sizeof(tlm_record_struct);
    msu_flash_read(device, msu_ptr, &etime, sizeof(DWORD));
    if (etime == 0xFFFFFFFFL)
        return(bottom | NO_TLM_WRAP);
    else if (etime == 0xFFFFFFFFL)
        return(top | NO_TLM_WRAP);

    /* ELSE -> not found via binary search... continue....*/

    /* Try a "bottom" based search.  That is, use a decreasing memory
     * binary search. */
    for (sector = 0; sector <= FLASH_SECTOR_MAX; sector++)
    {
        r = ((sector+1)*FLASH_SECTOR_SIZE)/sizeof(tlm_record_struct);
        msu_ptr = r * sizeof(tlm_record_struct);
        msu_flash_read(device, msu_ptr, &etime, sizeof(DWORD));
        if (etime != 0xFFFFFFFFL) /* in use, skip this sector */
            continue;
        else
        {
            /* Examine this sector for the first empty location */
            /* Do this with a binary search within this sector */
            bottom = (sector * FLASH_SECTOR_SIZE)/sizeof(tlm_record_struct);
            top = (((sector + 1) * FLASH_SECTOR_SIZE)/sizeof(tlm_record_struct)) - 1;
            while ((top - bottom) > 1)
            {
                mid = (bottom + top)/2;
                msu_ptr = mid*sizeof(tlm_record_struct);
                msu_flash_read(device, msu_ptr, &etime, sizeof(DWORD));
                if (etime != 0xFFFFFFFFL) /* in use, look higher */
                    bottom = mid;
                else
                    top = mid;
            }

            msu_ptr = bottom*sizeof(tlm_record_struct);
            msu_flash_read(device, msu_ptr, &etime, sizeof(DWORD));
            if (etime == 0xFFFFFFFFL)
                return(bottom | TLM_WRAP);
            else if (etime == 0xFFFFFFFFL)
                return(top | TLM_WRAP);
        }
    }

    return(NO_TLM_FIND);

}  /* End of msu_flash_search() */
```

```
/*****************************************************************************
 *
 *  int     msu_flash_erase_sector(int device, int sector)
 *
 *  Erases appropriate sector in the Flash array.  The sector is the absoulute
 *  sector for the entire array of Flash memory, thus the relative sector
 *  within the device must be determined, as well as the actual device itself.
 *
 *****************************************************************************/

int msu_flash_erase_sector(int device, int sector)
{
    register BYTE    fdata;
    register int pass = TRUE;
    DWORD            base;
    int              x;


    /* First, determine which of the Flash devices contains the absolute sector.
     * The compares are quicker than using arithmetic .
     */
    if (sector < FLASH_SECTORS_PER_DEVICE)
        base = 0L;
    else if (sector < FLASH_SECTORS_PER_DEVICE*2)
        base = FLASH_DEVICE_SIZE;
    else if (sector < FLASH_SECTORS_PER_DEVICE*3)
        base = FLASH_DEVICE_SIZE*2;
    else if (sector < FLASH_SECTORS_PER_DEVICE*4)
        base = FLASH_DEVICE_SIZE*3;

    msu_set_faddr(device, (base + 0x00005555L));
    pcb_write(device+1, 0, (BYTE)0xAA);
    msu_set_faddr(device, (base + 0x00002AAAL));
    pcb_write(device+1, 0, (BYTE)0x55);
    msu_set_faddr(device, (base + 0x00005555L));
    pcb_write(device+1, 0, (BYTE)0x80);
    msu_set_faddr(device, (base + 0x00005555L));
    pcb_write(device+1, 0, (BYTE)0xAA);
    msu_set_faddr(device, (base + 0x00002AAAL));
    pcb_write(device+1, 0, (BYTE)0x55);
    /* Relative sector within the Flash device in address bits A16 - A14. */
    msu_set_faddr(device, (base + (sector%FLASH_SECTORS_PER_DEVICE) << 14));
    pcb_write(device+1, 0, (BYTE)0x30);


    fdata = pcb_read(device+1, 0);
    x = 0;
    while (((fdata & 0x80) != 0x80) && (x < ERASE_TIME_LIMIT))
    {
        if (fdata & 0x20)
        {
            fdata = pcb_read(device+1, 0);
            if ((fdata & 0x80) == 0x80)
                break;
            else
            {
                pass = FALSE;
                break;
            }
        }

        fdata = pcb_read(device+1, 0);
        x++;

    }  /* End of WHILE */


    pcb_write(device, 2, 0x48);        /* Point to landing zone for low power */


    if (x == ERASE_TIME_LIMIT)
        return(0x80 | sector);
    else
        return(0);

}  /* End of msu_flash_erase_sector() */


/*****************************************************************************
 *
 *  int     msu_calc_first_rec(int rec_num)
 *
 *  Determine the first (oldest) record used in Flash based on the current
```

191

```
 *   record number and assuming data wrap around is in effect.
 *
 *   This location is the first complete (whole) record in the sector after
 *   the sector that contains the current record, rec_num.
 *
 ****************************************************************************/

int msu_calc_first_rec(int rec_num)
{
    int s, snext, r;


    /* Determine the next sector */
    s = (rec_num*sizeof(tlm_record_struct))/sizeof(FLASH_SECTOR_SIZE);
    if (s == FLASH_SECTOR_MAX)
        snext = 0;
    else
        snext = s + 1;

    /* determine # of complete records in Flash upto the next */
    r = (snext*sizeof(FLASH_SECTOR_SIZE))/sizeof(tlm_record_struct);

    /* the next complete record in the next sector is just the next record */

    return(r+1);

}  /* End of msu_calc_first_rec() */
```

End of msu.h, msu.c

pcb.h, pcb.c

```
/**************************************************************************
 *
 *   PCB.H
 *
 *   Include file for Peripheral Control Bus (PCB) software interface.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+------------------------------------------------
 *   4 March 1993 Jah      Creation
 *   8 Sept 1995     Jah      Adoption to PANSAT System Controller architecture
 *
 **************************************************************************/


#ifdef  PCB
    void            pcb_init(void);
    void            pcb_portc(int bitnum, int mode);
    unsigned int pcb_read(unsigned int select, unsigned int addr);
    void            pcb_write(unsigned int select, unsigned int addr, unsigned int value);
#endif

/* prototypes for modules other than pcb.c */
#ifndef PCB
    extern void            pcb_init(void);
    extern void            pcb_portc(int bitnum, int mode);
    extern unsigned int   pcb_read(unsigned int select, unsigned int addr);
    extern void            pcb_write(unsigned int select, unsigned int addr, unsigned int value);
#endif



/* Macros for PCBW and PCBR */
#define pcbw_m(select, addr, value) {\
    outp(PCB_PPI_PORTA, (value)); \
    outp(PCB_PPI_PORTB, PCB_READ_OFF  | PCB_WRITE_OFF      | ((select) & 0x000F) | (((addr) & 0x0003) << 4) );
\
    outp(PCB_PPI_PORTB, PCB_WRITE_ON  | ((select) & 0x000F) | (((addr) & 0x0003) << 4) ); \
    outp(PCB_PPI_PORTB, PCB_WRITE_OFF | ((select) & 0x000F) | (((addr) & 0x0003) << 4) ); }

#define pcbr_m(select, addr, value) {\
    outp(PCB_PPI_PORTB, PCB_READ_OFF | PCB_WRITE_OFF      | ((select) & 0x000F) | (((addr) & 0x0003) << 4) );
\
    outp(PCB_PPI_PORTB, PCB_READ_ON  | ((select) & 0x000F) | (((addr) & 0x0003) << 4) ); \
    outp(PCB_PPI_CTRL, 2);\
    outp(PCB_PPI_CTRL, 3);\
    outp(PCB_PPI_PORTB, PCB_READ_OFF | ((select) & 0x000F) | (((addr) & 0x0003) << 4) ); \
    (value) = inp(PCB_PPI_PORTA); }


/**************************************************************
 *
 *      PPI Interface on a DCS to control the Peripheral Control Bus.
 *
 *      Using Mode 2 (0xC0) the PPI is programmed to support strobed
 *      bidirectional bus I/O using Port A.  Port B is used as output, and
 *      port C is used for handshaking and other output purposes.
 *
 *      Port A contains the data (byte value) which is moved across the
 *      control bus  Port B contains the device selection and sub-address
 *      control bits, and the Read and Write strobes.
 *
 **************************************************************/

#define PCB_PPI_BASE     0x100
#define PCB_PPI_PORTA    PCB_PPI_BASE+0
#define PCB_PPI_PORTB    PCB_PPI_BASE+2
#define PCB_PPI_PORTC    PCB_PPI_BASE+4
#define PCB_PPI_CTRL     PCB_PPI_BASE+6

#define PCB_PPI_INIT     0xC0

/* Read and Write Line Toggling Controls */
#define PCB_READ_ON      0x40
#define PCB_READ_OFF     0xC0
```

```
#define PCB_WRITE_ON    0x80
#define PCB_WRITE_OFF   0xC0



/******************************************************************
 ******************************************************************
 *
 * Devices on the PCB
 *
 * These devices are selected via the DCS PPI for PCB control.
 * PPI port B is the addressing register.  The following device
 * selects use bits 3 - bits 0 (D3 - D0) of PPI Port B.
 *
 * Note that the low order bit (D0) is used to differentiate
 * between the two selects on the same unit.  Device subaddresses
 * are not included here, since they are particular to a device.
 *
 ******************************************************************/

#define SCA       0x02
#define SCA0      SCA       /* System Control A */
#define SCA1      (SCA+1)

#define SCB       0x0A
#define SCB0      SCB       /* System Control B */
#define SCB1      (SCB+1)

#define TMUXA     0x04
#define TMUXA0    TMUXA     /* Analog MUX A */
#define TMUXA1    (TMUXA+1)

#define TMUXB     0x0C
#define TMUXB0    TMUXB     /* Analog MUX B */
#define TMUXB1    (TMUXB+1)

#define MSA       0x06
#define MSA0      MSA       /* Mass Storage A */
#define MSA1      (MSA+1)

#define MSB       0x0E
#define MSB0      MSB       /* Mass Storage B */
#define MSB1      (MSB+1)

#define RF        0x00
#define RF0       RF        /* RF System */
#define RF1       (RF+1)

#define EPS       0x08
#define EPS0      EPS       /* Electrical Power System */
#define EPS1      (EPS+1)

/* EPS Selects S3-S0 */
#define EPS_PORT_S0     0x08
#define EPS_PORT_S1     0x08
#define EPS_PORT_S2     0x08
#define EPS_PORT_S3     0x08
#define EPS_PORT_S4     0x09
#define EPS_PORT_S5     0x09
#define EPS_PORT_S6     0x09
#define EPS_PORT_S7     0x09

/* EPS Sub-addresses A1-A0 */
#define EPS_PORT_A0     0x00
#define EPS_PORT_A1     0x01
#define EPS_PORT_A2     0x02
#define EPS_PORT_A3     0x03
#define EPS_PORT_A4     0x00
#define EPS_PORT_A5     0x01
#define EPS_PORT_A6     0x02
#define EPS_PORT_A7     0x03

/******************************************************************
 ******************************************************************
 *
 * Subsystem addresses of PCB devices
 *
 ******************************************************************/

/******************************************************************
 * Mass Storage
 * PPI Interface: Indexed using MSx1
 * Data Port:     Indexed using MSx2
 ******************************************************************/
```

194

```
#define MS_PPI_BASE              0
#define MS_PPI_PORTA    MS_PPI_BASE+0
#define MS_PPI_PORTB    MS_PPI_BASE+1
#define MS_PPI_PORTC    MS_PPI_BASE+2
#define MS_PPI_CTRL             MS_PPI_BASE+3


/******************************************************************************
 *
 *   PCB.C
 *
 *   Interface routines for the Peripheral Control Bus (PCB).
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who       What
 *   ---------------+-------+---------------------------------------------
 *   4 March 1993 Jah      Creation
 *   8 Sept 1995      Jah      Adoption to PANSAT System Controller architecture
 *   24 April 1996    Jah      Make PCB functions non-interruptable.
 *
 ******************************************************************************/

#include        "gen_defs.h"
#include        "gen_apis.h"

#define          PCB
#include        "pcb.h"
#undef           PCB

#include        "dcs.h"



/******************************************************************************
 *
 *   pcb_init()
 *
 *   Initializes the PCB by preparing the PPI controlling the PCB to work
 *   in the bidirectional strobed data mode, and to make sure that no
 *   read or write commands are occuring on the PCB.
 *
 ******************************************************************************/

void     pcb_init(void)
{
    outp(PCB_PPI_CTRL, PCB_PPI_INIT);

    outp(PCB_PPI_PORTB, 0xC0);

    pcb_portc(0, OFF);              /* Modem Power OFF */
    pcb_portc(1, SET);             /* PC1 = PPI Input Strobe ON */
    pcb_portc(2, RESET);        /* Reset EDAC Error Acknowledge */
    pcb_portc(2, SET);

}  /* End of pcb_init() */


/******************************************************************************
 *
 *   pcb_portc()
 *
 *   Toggles the three output bits of Port C on the PPI.
 *
 ******************************************************************************/

void     pcb_portc(int bitnum, int mode)
{
    register unsigned char    temp;
    register WORD             state;


    mode &= 0x01;            /* make sure only bit D0 is used in mode */
```

195

```c
        state = disable_ints();

        switch(bitnum)
        {
            case 0:
                temp = mode;
                break;

            case 1:
                temp = 0x02 | mode;
                break;

            case 2:
                temp = 0x04 | mode;
                break;

            default:
                temp = 0xFF;
        }

        if (temp != 0xFF)
            outp(PCB_PPI_CTRL, temp);

        if (state)
            enable_ints();

}   /* End of pcb_portc() */


/********************************************************************
 *
 *  pcb_read()
 *
 *  Read one data (byte) from the Peripheral Control Bus via the onboard PPI.
 *
 ********************************************************************/

WORD    pcb_read(unsigned int select, unsigned int addr)
{
    /* Note: register not used so that this routine is re-entrable
     *    at the point of the last statement, return(temp).
     */
    register WORD   temp = (select & 0x000F) | ((addr & 0x0003) << 4);
    WORD            value;
    register WORD      state;


    state = disable_ints();

     /* Set Device Select and address without read or write commands */
    outp(PCB_PPI_PORTB, 0xC0 | temp);

    outp(PCB_PPI_PORTB, 0x40 | temp);  /* PB7 = /RD goes LOW */
    outp(PCB_PPI_CTRL, 2);                   /* PC1 = /Strobe goes LOW */
    outp(PCB_PPI_CTRL, 3);                   /* PC1 = /Strobe goes HIGH */
    outp(PCB_PPI_PORTB, 0xC0 | temp);  /* PB7 = /RD goes HIGH */

    value = inp(PCB_PPI_PORTA);

    if (state)
        enable_ints();

    /* Note: enabling interrupts BEFORE returning from PCBR can result
     *  in another piece of software (e.g. inside an ISR) calling PCBR
     *      during the stack manipulations following the above _enable().
     *  This would actually result in the issuing of the new PCBR and
     *  returning the value of the PCBR, followed by that piece of software
     *  returning via an IRET, and then the PCBR that was interrupted would
     *  coutinue with its return(...).
     */

    return(value);

}   /* End pcb_read() */


/********************************************************************
 *
 *  pcb_write()
 *
 *  Write one data (byte) from the Peripheral Control Bus via the onboard PPI.
 *
 ********************************************************************/
```

196

```
void     pcb_write(unsigned int select, unsigned int addr, unsigned int value)
{
    register WORDtemp = (select & 0x000F) | ((addr & 0x0003) << 4);
    register WORD     state;

    state = disable_ints();

    outp(PCB_PPI_PORTA, value);

    /* Set Device Select and address without read or write commands */
    outp(PCB_PPI_PORTB, (PCB_READ_ON | PCB_WRITE_ON) | temp);

    /* Toggle the Write line */
    outp(PCB_PPI_PORTB, (PCB_WRITE_ON | temp));
    outp(PCB_PPI_PORTB, (PCB_WRITE_OFF | temp));

    if (state)
        enable_ints();

}   /* End pcb_write() */
```

End of pcb.h, pcb.c

print.h, print.c

```
/**********************************************************************
 *
 *      PRINT.H
 *
 *      DCS printf:     void dprint(char *format,...)
 *
 *      Petite Amateur Navy Satellite (PANSAT).
 *      Embedded ROM software.
 *      Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *      Jim A. Horning (Jah)
 *
 *      Revision History:
 *      =================
 *
 *      Date            Who     What
 *      ---------------+-------+-------------------------------------------
 *      1 Feb 1991      Jah     Creation (Star)
 *      2 Nov 1993              Jah             Adopted for DCS (from Star)
 *
 **********************************************************************/

/* Include specifics for PRINT.C */
#ifdef   PRINT
     /* Internal routines to print.c */
     static int      get_width(char *, int *);
     static int      get_precision(char *, int *);
     static void     print_fp(char *obuf, double x, int precision, int width);
     static void     print_ptr(char *, void far *, int);
     static unsigned long int        power(unsigned int x, unsigned int y);
#endif

/* Include for all other modules */
#ifndef PRINT
     extern void     dprint(char *format,...);
#endif

/**********************************************************************
 *
 *   PRINT.C
 *
 *   DCS printf:  void dprint(char *format,...)
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+-------------------------------------------
 *   1 Feb 1991      Jah     Creation
 *   2 Nov 1993      Jah     Adopted for DCS
 *   18 Apr 1996     Jah     FP support
 *
 **********************************************************************/


#include <stdarg.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>


#include "gen_defs.h"

#define     PRINT
#include "print.h"
#undef      PRINT

#include "scc.h"


#define NEGATIVE        1
#define PLUS            2
#define NEAR_PTR        1
#define FAR_PTR         2


/**********************************************************************
```

```
 *
 *      get_width()
 *
 *      Get the width for a specific output parameter.  This is a number field
 *      placed in front of the format type of a star_print format.  This
 *      number is either followed by a format type (letter) or a period (.).
 *      The period denotes that a precision field will also be specified.
 *
 *      This routine outputs the width.
 *
 *****************************************************************************/

static int   get_width(char *format, int *i)
{
        int     itemp;
        char    buf[10];
        int     n;


        itemp = *i;
        n = 0;
        buf[n] = NULL_CHAR;
        while (isdigit(format[itemp]))
                buf[n++] = format[itemp++];
        buf[n] = NULL_CHAR;

        *i = itemp;             /* update the marker in format string */

        return(atoi(buf));

}   /* End of get_width() */


/*****************************************************************************
 *
 *      get_precision()
 *
 *      Get the precision for a specific output parameter.  This is a number
 *      field which can be preceeded by a + (default) or -.  This number is
 *      then followed by a format type (letter).
 *
 *      See print_fp() for more details.
 *
 *      This routine outputs the precision which can be positive or negative.
 *
 *****************************************************************************/

static int   get_precision(char *format, int *i)
{
        int     itemp;
        char    buf[10];
        int     n;
        int     sign;
        int     precision;


        itemp = *i;
        sign = PLUS;
        if (format[itemp] == '-')
        {
                sign = NEGATIVE;
                itemp++;
        }
        else if (format[itemp] == '+')
                itemp++;        /* already is PLUS don't need to flag this */

        n = 0;
        buf[n] = NULL_CHAR;
        while (isdigit(format[itemp]))
                buf[n++] = format[itemp++];
        buf[n] = NULL_CHAR;

        precision = atoi(buf);
        if (sign == NEGATIVE)
                precision *= -1;
        *i = itemp;                     /* update the marker in format string */

        return(precision);

}   /* End of get_precision() */


/*****************************************************************************
```

```
 *
 *        print_fp()
 *
 *        Display value of a floating point number, either float or double.
 *        (The float is already casted as a double when passed to this routine.
 *
 *************************************************************************/

static void  print_fp(char *obuf, double x, int precision, int width)
{
        char      buf[40];
        int       buf_cnt = 0;
        double       q, q1, q2;
        long int xi, d;
        int          c, n;


     gcvt(x, precision, buf);
     strcat(obuf, buf);
     return;


     if (x == 0.0)
     {
         buf[buf_cnt++] = '0';
         buf[buf_cnt++] = '.';
         buf[buf_cnt++] = '0';
         buf[buf_cnt] = NULL_CHAR;
         strcat(obuf, buf);
         return;
     }


     /* Adjust precision to correct defaults for FP printing */
     if (precision > 8)
         precision = 8;
     if (precision < 0)
         precision = 0;


     if (x < 0.0)
     {
         buf[buf_cnt++] = '-';
         x *= -1;
     }

     /* Determine # of digits BEFORE the decimal place */
     for (c = 0, q = x; q > 1; c++)
         q /= 10.0;

     if (c == 0)                        /* add leading zero before . -> 0.xyz */
         buf[buf_cnt++] = '0';

     /* Display the digits BEFORE the decimal place */
     for (q = x, n = 0; n < c; n++)
     {
         d = power(10, (c-1)-n);
         q1 = q/d;
         buf[buf_cnt++] = '0' + (unsigned int)q1;   /* TOP digit */

         q2 = (unsigned int)q1 * d;         /* Only the most significant */
         q = q - q2;                        /* Remove largest factor of 10 */
     }
     buf[buf_cnt++] = '.';

     for (n = 0; n < precision; n++)
     {
         q = q * 10;
         q1 = (unsigned int) q;          /* 0.xyz --> x.yz */
         buf[buf_cnt++] = '0' + q1;      /* TOP digit (x) */

         q = q - q1;                     /* remove largest factor of 10 */
     }

     buf[buf_cnt++] = NULL_CHAR;
     strcat(obuf, buf);
     return;

}  /* End of print_fp() */


/*************************************************************************
 *
 *        print_ptr()
```

200

```
 *
 *          Pointer print for FAR pointer (SEGMENT:OFFSET) or
 *          NEAR pointer (OFFSET).
 *
 *          The parameter mode indicates if this is for NEAR or FAR pointer
 *          printing.
 *
 *          The format for output is:
 *                                      0         1
 *                  (char position) 01234567890
 *                  NEAR            xxxx             <-- the OFFSET
 *                  FAR             xxxx:yyyy        <-- the OFFSET:SEGMENT
 *
 **************************************************************************/

static void  print_ptr(char *obuf, void far *ptr, int mode)
{
        int             n;
        unsigned char   c;
        char            buf[10];
        int             b_cnt;
        unsigned long   p;



        if (mode == FAR_PTR)
                b_cnt = 9;
        else
                b_cnt = 4;
        p = (unsigned long int) ptr;
        buf[b_cnt--] = NULL_CHAR;
        for (; b_cnt >= 0; b_cnt--)
        {
                c = (unsigned char) (0x0000000FL & p);
                if ((c >= 0) && (c <= 9))
                        buf[b_cnt] = c + '0';
                else
                        buf[b_cnt] = (c - 0x0A) + 'A';
                p >>= 4;
                /* p>>=1; p>>=1; p>>=1; p>>=1; */
                if (b_cnt == 5)
                        buf[--b_cnt] = ':';
        }

        strcat(obuf, buf);

}   /* End of print_ptr() */


/**************************************************************************
 *
 *          dprint()
 *
 *          Printf for DCS.
 *
 *          Special characters (translated by compiler!)
 *          "\n"      CRLF
 *          "\\"      \
 *          "\%"      %
 *
 *          Format: %[width][.precision][size]type
 *
 *          Types
 *          %c        char
 *          %s        string (char *)
 *          %d        int (as decimal)
 *          %i        int
 *          %u        unsigned int
 *          %x        int (as hex)
 *          %X        int (as HEX)
 *          %p        NEAR pointer (OFFSET)
 *          %P        FAR pointer (SEGMENT:OFFSET)
 *          %f        fp (%lf -> double, %f ->float)
 *
 *          size
 *          L/l - long
 *          H/h - short
 *
 **************************************************************************/

void    dprint(char *format,...)
{
    va_list arg_ptr;                    /* ptr to argument list */
    int     i;                          /* index in format string */
```

```c
int     temp_int;
long    temp_long;
double  temp_fp;
char    temp_char;
char    *temp_charptr;
void    far *temp_ptr;
void    *temp_ptr_near;
static  char buf[20];           /* temp buffer */
static  char obuf[200];         /* the output buffer */
int     hflag, lflag;
int     hex_upper;
int     width, precision;
static  char    crlf[3] = {CR, LF, NULL_CHAR};
int     n;


/* start variable arguement fetching */
va_start(arg_ptr, format);

i = 0;
obuf[0] = NULL_CHAR;
while (format[i] != NULL_CHAR)
{
    hex_upper = hflag = lflag = FALSE;
    width = 100;
    precision = 3;
    buf[0] = NULL_CHAR;
    switch (format[i])
    {
        case CR:
        case LF:
            i++;
            strcat(obuf, crlf);
            break;

        case '%':
            i++;

            /* width ? */
            if ((format[i] >= '0') && (format[i] <= '9'))
            {
                /* i++; */     /* or more */
                width = get_width(format, &i);
            }

            /* .precision ? */
            if (format[i] == '.')
            {
                i++;    /* or more */
                precision = get_precision(format, &i);
            }

            if ((format[i] == 'h') || (format[i] == 'H'))
            {
                i++;
                hflag = TRUE;
                /* set h flag */
            }

            if ((format[i] == 'l') || (format[i] == 'L'))
            {
                i++;
                lflag = TRUE;
                /* set l flag */
            }

            switch(format[i])/* TYPE */
            {
                case 'd':
                case 'D':
                case 'i':
                case 'I':
                    i++;
                    if (lflag)
                        temp_long = va_arg(arg_ptr, long);
                    else
                        temp_long = (long)va_arg(arg_ptr, int);
                    ltoa(temp_long, buf, 10);
                    strncat(obuf, buf, width);
                    break;

                case 'u':
                case 'U':
                    i++;
```

```
                        if (lflag)
                            temp_long = va_arg(arg_ptr, long);
                        else
                            temp_long = (long)va_arg(arg_ptr, int);
                        ltoa(temp_long, buf, 10);
                        strncat(obuf, buf, width);
                        break;

                    case 'X':
                        hex_upper = TRUE;
                    case 'x':
                        i++;
                        if (lflag)
                            temp_long = va_arg(arg_ptr, long);
                        else
                            temp_long = 0x0000FFFFL&(long)va_arg(arg_ptr, int);
                        ltoa(temp_long, buf, 16);
                        if (hex_upper)
                            for (n = 0; buf[n] != NULL_CHAR; n++)
                                buf[n] = (char)toupper(buf[n]);
                        strncat(obuf, buf, width);
                        break;

                    case 'c':
                    case 'C':
                        i++;
                        temp_int = va_arg(arg_ptr, int);
                        temp_char = (char)(temp_int & 0x00FF);
                        buf[0] = temp_char;
                        buf[1] = NULL_CHAR;
                        strcat(obuf, buf);
                        break;

                    case 's':
                    case 'S':
                        i++;
                        temp_charptr = va_arg(arg_ptr, char *);
                        strcat(obuf, temp_charptr);
                        break;

                    case 'f':
                    case 'F':
                        i++;
                        if (lflag)
                            temp_fp = va_arg(arg_ptr, double);
                        else
                            temp_fp = (double)va_arg(arg_ptr, float);
                        print_fp(obuf, temp_fp, precision, width);
                        break;

                    case 'p':
                        i++;
                        temp_ptr = (void far *)va_arg(arg_ptr, void *);
                        print_ptr(obuf, temp_ptr, NEAR_PTR);
                        break;

                    case 'P':
                        i++;
                        temp_ptr = va_arg(arg_ptr, void far *);
                        print_ptr(obuf, temp_ptr, FAR_PTR);
                        break;

                    default:    /* not supported */
                        i++;
                        break;

                }   /* End of SWITCH */

                break;  /* End of CASE '%' */

            default:
                strncat(obuf, (format+i), 1);
                i++;
                break;

        }   /* End of SWITCH */

    }   /* End of WHILE */

    /* send to serial port */
    put_string(obuf);

    va_end(arg_ptr);
```

```
}   /* End of dprint() */


/*****************************************************************************
 *
 *   pow()
 *
 *****************************************************************************/

unsigned long intpower(unsigned int x, unsigned int y)
{
    unsigned int    i;
    unsigned long intp;

    p = 1;
    for (i = 1; i <= y; i++)
        p *= x;

    return(p);

}   /* End of pow() */
```

End of print.h, print.c

rf.h, rf.c

```
/****************************************************************************
 *
 *   RF.H
 *
 *   Defines for the RF unit interface routines.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   ==================
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   30 Oct 1996     Jah     Creation
 *
 ****************************************************************************/

#define RF_HPA        7
#define RF_LNA        6
#define RF_LHP_LHA    3
#define RF_LOP_LOA    2
#define RF_TX_RX 1
#define RF_T_R        0


#ifdef RF
        #define  T0CON    0xFF56
        #define  T0CNT    0xFF50
        #define  T0CMPA   0xFF52
        #define  T0CMPB   0xFF54

        #define  T1CON    0xFF5E
        #define  T1CNT    0xFF58
        #define  T1CMPA   0xFF5A
        #define  T1CMPB   0xFF5C


        static void  rf_power(int mode);
        static void  rf_set(int ctrl, int mode);
        static void  rf_timer(int delay);
        static void  rf_txpower(int mode);

#endif


#ifndef RF
        extern void  rf_power(int mode);
        extern void  rf_set(int ctrl, int mode);
        extern void  rf_timer(int delay);
        extern void  rf_txpower(int mode);

#endif

/****************************************************************************
 *
 *   RF.C
 *
 *   Interface routines for the RF unit.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   ==================
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   30 Oct 1996     Jah     Creation
 *
 ****************************************************************************/

#include     "gen_defs.h"

#define      RF
#include "rf.h"
#undef       RF
```

```c
static int   rf_pcb = 0;


/*****************************************************************************
 *
 *   void rf_power(int mode)
 *
 *   Used to turn on or off the power to the entire RF unit.
 *
 *****************************************************************************/

void rf_power(int mode)
{
    eps_set_power(RF, mode);

}   /* End of rf_power() */


/*****************************************************************************
 *
 *   void rf_set(int ctrl, int mode)
 *
 *   Allows individual bit control of the RF PCB controls.  Any bit can be
 *   toggled preserving other control bits.  Note: the LNA logic is reversed.
 *
 *****************************************************************************/

void rf_set(int ctrl, int mode)
{
    switch(ctrl)
    {
        case RF_T_R:
        case RF_TX_RX:
        case RF_LOP_LOA:
        case RF_LHP_LHA:
        case RF_HPA:
            if (mode == ON)
                rf_pcb |= (1 << ctrl);
            else
                rf_pcb &= ~(1 << ctrl);
            pcb_write(RF, 0, rf_pcb);
            break;

        case RF_LNA:
            /* reversed logic for LNA control: this is because when the RF
             * unit goes on, you want an LNA on by default.
             */
            if (mode == OFF)
                rf_pcb |= (1 << ctrl);
            else
                rf_pcb &= ~(1 << ctrl);
            pcb_write(RF, 0, rf_pcb);
            break;

        default:
            break;
    }

}   /* End of rf_set() */


/*****************************************************************************
 *
 *   void rf_timer(int delay)
 *
 *   Begins the RF transmitter timer using delay as a parameter (in seconds).
 *   Timers  0 and 1 are used in the cascade mode.  Timer 0 is set to a maximum
 *   count, internal clock, retrigger, using Compare A only.  Timer 1 is set
 *   to use an external clock (output of Timer 0), in a one shot mode, using
 *   the dual mode Compare A/Compare B.
 *
 *****************************************************************************/

void rf_timer(int delay)
{
    outpw(T0CNT, 0);
    outpw(T0CMPA, 0);          /* maximum count (65536) */
    outpw(T0CON, 0xC001);      /* internal clk, retrigger, CMPA only */

    outpw(T1CNT, 0);
    outpw(T1CMPA, 1);          /* smallest compare A */

    outpw(T1CMPB, delay * 28);    /* ~28 CMPB per second */
    outpw(T1CON, 0xC006);      /* ext. clk, 1 shot, CMPA/CMPB dual mode */
```

206

```
}   /* End of rf_timer() */


/*********************************************************************
 *
 *   void rf_txpower(int mode)
 *
 *   Change power level of the transmitter by controlling the 2-bit
 *   attenuator.
 *
 *********************************************************************/

void rf_txpower(int mode)
{
    rf_pcb ~= 0xCF;                    /* mask off all power bits (set to 0) */
    rf_pcb |= ((mode & 0x03) << 4);
    pcb_write(RF, 0, rf_pcb);

}   /* End of rf_txpower() */
```

End of rf.h, rf.c

scc.h, scc.c

```
/*****************************************************************************
 *
 *    SCC.H
 *
 *    SCC
 *
 *         Petite Amateur Navy Satellite (PANSAT).
 *         Embedded ROM software.
 *         Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *         Jim A. Horning (Jah)
 *
 *         Revision History:
 *         =================
 *    Date             Who      What
 *    ---------------+-------+---------------------------------------------
 *         17 July 1996    Jah            Creation
 *
 *****************************************************************************/

typedef struct
{
    BYTE     reg;
    BYTE     data;

} scc_instr_struct;


#define SCCA             0
#define SCCB             1

#define SCC_CHA_BUF_SIZE     516
#define SCC_CHB_BUF_SIZE     2048

#define SCCA_CMD      2                          /* Channel A Command */
#define SCCA_DATA     6                          /* Channel A Data */
#define SCCB_CMD      0                          /* Channel B Command */
#define SCCB_DATA     4                          /* Channel B Data */


#ifdef   SCC


    #define CHA_BUF_SIZE     SCC_CHA_BUF_SIZE
    #define CHB_BUF_SIZE     SCC_CHB_BUF_SIZE

    unsigned          cnv_hex(char buf[]);
    unsigned long int cnv_lhex(char buf[]);
    char              get_char(void);
    void              get_string(char *string, int max);
    void              put_string(char *string);
    void              hex_ascii_dump(BYTE *ptr, int count);
    void              scc_init(void);
    void              scc_write_table(scc_instr_struct table[], int channel);
    BYTE              serial_in(void);
    void              serial_out(BYTE c);
    void              scca_wreg(int reg, int value);
    void              sccb_wreg(int reg, int value);
    void              scc_hunt(void);

#endif


#ifndef SCC

    extern unsigned cnv_hex(char buf[]);
    extern unsigned long int cnv_lhex(char buf[]);
    extern char           get_char(void);
    extern void           get_string(char *string, int max);
    extern void           put_string(char *string);
    extern void           hex_ascii_dump(BYTE *ptr, int count);
    extern void           scc_init(void);
    extern void           scc_write_table(scc_instr_struct table[], int channel);
    extern BYTE           serial_in(void);
    extern void           serial_out(BYTE c);

    extern void           scca_wreg(int reg, int value);
    extern void           sccb_wreg(int reg, int value);
    extern void           scc_hunt(void);

    extern BYTE           cha_in_buf0[];
    extern BYTE           cha_out_buf0[];
```

```c
        extern BYTE             txa;
        extern BYTE             rx_eom;
        extern BYTE             hunt;
        extern WORD             a_txunderrun_eom;
        extern WORD             a_rxoverrun;
        extern WORD             a_brk_abort;

#endif

/****************************************************************************
 *
 *   SCC.C
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *   17 July 1996 Jah       Creation
 *
 ****************************************************************************/

#include         "gen_defs.h"
#include         "gen_apis.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define          SCC
#include         "scc.h"
#undef           SCC

#include         "dcs.h"
#include         "print.h"
#include         "terms.h"



WORD jim = 0;
WORD b1 = 0;
WORD b2 = 0;
WORD b3 = 0;
WORD b4 = 0;
WORD a1 = 0;
WORD a2 = 0;
WORD a3 = 0;
WORD a4 = 0;

scc_instr_struct scca_init[] =
{
    {0x09,   0xC0},   /* force hardware reset */
    {0x04,   0x20},   /* x1 clock, SDLC mode, SYNC mode */
    {0x01,   0x00},   /* disable DMA and interrupts */
    {0x02,   0x00},   /* zero interrupt vectors */
    {0x03,   0xC8},   /* Rx 8 bits, Rx CRC enabled */
    {0x05,   0x61},   /* Tx 8 bits, Tx CRC enabled */
    {0x06,   0x00},   /* SDLC address field */
    {0x07,   0x7E},   /* AUTO RTS, EOM, TX flag */
    {0x09,   0x22},   /* No vector and no INTACK */
    {0x0A,   0xA0},   /* CRC preset=1, NRZI, Flag idle */
    {0x0B,   0x09},   /* TxCK=TRxC, RxCK=RTxC  */
    {0x0C,   0x2E},   /* TCL 76,800 Hz */
    {0x0D,   0x00},   /* TCH */
    {0x0E,   0x02},   /* BRG=PCLK, DPPL=BRG */
    {0x0E,   0x03},   /* Enable BRG, BRG=PCLK  */
    {0x03 ,  0xC9},   /* Enable RX, CRC */
    {0x05 ,  0x69},   /* Enable TX, CRC */
    {0x00 ,  0x80},   /* Reset TxCRC */
    {0x0f ,  0x00},   /* clear all IE bits */
    {0x00 ,  0x10},   /* reset ext/status */
    {0x00 ,  0x10},   /* reset ext/status */
    {0x01 ,  0x10},   /* int on all Rx or special */
    {0x09 ,  0x2A},   /* enable interrupts}; */
    {0xFF,   0xFF}    /* End */
};


scc_instr_struct sccb_init[] =
```

```
{
    {0x04,    0x44},    /* x16 clock, 1 stop, no parity */
    {0x03,    0xC0},    /* Rx 8 bits, Rx disabled */
    {0x05,    0x60},    /* Tx 8 bits, DTR, RTS, Tx off */
    {0x0B,    0x56},    /* Tx & Rx=BRG, TRxC=BRG */
    {0x0C,    0x16},    /* TCL */
    {0x0D,    0x00},    /* TCH */
    {0x0E,    0x02},    /*    */
    {0x0E,    0x03},    /*    */
    {0x03,    0xC1},    /* Rx 8 bits, Rx enabled */
    {0x05,    0x6A},    /* Tx 8 bits, Tx, RTS enabled */
    {0xFF,    0xFF}     /* End */
};



BYTE hunt = FALSE;
WORD a_txunderrun_eom = 0;
WORD a_rxoverrun = 0;
WORD a_brk_abort = 0;

BYTE cha_in_buf0[CHA_BUF_SIZE];
BYTE cha_out_buf0[CHA_BUF_SIZE];
BYTE txa = OFF;
BYTE rx_eom = FALSE;

BYTE chb_in_buf[CHB_BUF_SIZE];
BYTE chb_out_buf[CHB_BUF_SIZE];
int      chb_in_now = 0;
int      chb_in_next = 0;
int      chb_out_now = 0;
int      chb_out_next = 0;



/*****************************************************************************
 *
 *   cnv_hex()
 *
 *****************************************************************************/

unsigned cnv_hex(char buf[])
{
    int      i;
    unsigned int val = 0;



    i = 0;
    while (buf[i] != '\0')
    {
        val *= 16;

/*        ** ERROR ** w/ MSC6.0 compiler run-time lib*/
/*        if (isdigit(buf[i])) */
/*        ** ERROR ** w/ MSC6.0 compiler run-time lib*/

        if ((buf[i] >= '0') && (buf[i] <= '9'))
            val += buf[i] - '0';

        else
            val += toupper(buf[i]) - 'A' + 0x0A;

        i++;
    }

    return(val);

}  /* End of cnv_hex() */


/*****************************************************************************
 *
 *   cnv_lhex()
 *
 *****************************************************************************/

unsigned long int cnv_lhex(char buf[])
{
    int      i;
    unsigned long int val = 0;
```

210

```
        i = 0;
        while (buf[i] != '\0')
        {
            val *= 16;

/*          ** ERROR ** w/ MSC6.0 compiler run-time lib*/
/*          if (isdigit(buf[i])) */
/*          ** ERROR ** w/ MSC6.0 compiler run-time lib*/

            if ((buf[i] >= '0') && (buf[i] <= '9'))
                val += buf[i] - '0';

            else
                val += toupper(buf[i]) - 'A' + 0x0A;

            i++;
        }

        return(val);

}   /* End of cnv_lhex() */


/*****************************************************************************
 *
 *  get_char()
 *
 *****************************************************************************/

char get_char(void)
{
    while (!is_serial_in())
        ;

    return(serial_in());

}   /* End of get_char() */


/*****************************************************************************
 *
 *  get_string()
 *
 *  Get a character string delimited by an ENTER and null-terminate
 *  it.  Can only accept upto max characters.
 *
 *****************************************************************************/

void get_string(char *string, int max)
{
    register char   c;
    register int i;


    i = 0;
    c = get_char();
    while (c != CR)
    {
        if (c == BACK_SPACE)
        {
            if (i > 0)
            {
                i--;
                serial_out((BYTE)BACK_SPACE);
                serial_out((BYTE)' ');
                serial_out((BYTE)BACK_SPACE);

            }
        }

        else if (i < max)
        {
            string[i++] = c;
            serial_out((BYTE)c);
        }

        c = get_char();

    }   /* End of WHILE */

    string[i] = NULL_CHAR;          /* Note: CR not saved - NULL instead */

}   /* End of get_string() */
```

211

```c
/******************************************************************
 *
 *   hex_ascii_dump()
 *
 ******************************************************************/

void    hex_ascii_dump(BYTE *ptr, int count)
{
    WORD    val;
    int     i, j;


    for (i = 0; i < count; i += 16)
    {
        for (j = 0; j < 16; j++)
        {
            val = ((WORD)*(ptr+i+j) & 0x00FF);
            if (val < 0x10)
                dprint("0");
            dprint("%X ", val);
        }
        dprint("  ");
        for (j = 0; j < 16; j++)
        {
            val = ((WORD)*(ptr+i+j) & 0x00FF);
            if ((val >= 32) && (val <= 127))
                dprint("%c",*(ptr+i+j));
            else
                dprint(".");
        }
        dprint("\n");
    }

}   /* End of hex_ascii_dump() */


/******************************************************************
 *
 *   put_string()
 *
 *   Send a null-terminated character string to the serial port.
 *
 ******************************************************************/

void put_string(char *string)
{
    register WORD    state;


    while (*string != NULL_CHAR)
    {
        chb_out_buf[chb_out_next++] = *string++;
        if (chb_out_next > CHB_BUF_SIZE-1)
            chb_out_next = 0;
    }

    state = disable_ints();
    if (inp(SCCB_CMD) & 0x04)
    {
        outp(SCCB_DATA, chb_out_buf[chb_out_now++]);

        if (chb_out_now > CHB_BUF_SIZE-1)
            chb_out_now = 0;
    }

    if (state)
        enable_ints();

}   /* End of put_string() */


/******************************************************************
 *
 *   scc_init()
 *
 ******************************************************************/

void scc_init(void)
{
    scc_write_table(scca_init, SCCA);
    scc_write_table(sccb_init, SCCB);

}   /* End of scc_init() */
```

```c
/**************************************************************************
 *
 *   scc_write_table()
 *
 **************************************************************************/

void scc_write_table(scc_instr_struct table[], int channel)
{
    int     x;
    int     port;


    port = (channel == SCCA) ? SCCA_CMD : SCCB_CMD;

    inp(port);          /* read status */

    for (x = 0; table[x].reg != (BYTE)0xFF; x++)
    {
        outp(port, table[x].reg);
        outp(port, table[x].data);
    }

}   /* End of scc_write_table() */


/**************************************************************************
 *
 *   serial_in()
 *
 *   Serial input.  Check to see if kbhit() was called to see if there
 *   was a key pressed (and thus the key pressed has already been
 *   retrieved and saved), if so return this.  Otherwise, return 0.
 *
 **************************************************************************/

BYTE    serial_in(void)
{
    register BYTE    key;


    if (chb_in_now == chb_in_next)
        key = 0;

    else
    {
        key = chb_in_buf[chb_in_now++];
        if (chb_in_now > CHB_BUF_SIZE-1)
            chb_in_now = 0;
    }

    return(key);

}   /* End of serial_in() */


/**************************************************************************
 *
 *   is_serial_in()
 *
 **************************************************************************/

int is_serial_in(void)
{
    if (chb_in_now == chb_in_next)
        return(FALSE);

    else
        return(TRUE);

}   /* End of is_serial_in() */


/**************************************************************************
 *
 *   serial_out()
 *
 *   Serial output.  Add the given character to the wrap-around output buffer.
 *   No check is made to see if the buffer is full.
 *
 **************************************************************************/

void    serial_out(BYTE c)
```

```c
{
    register WORD    state;


    chb_out_buf[chb_out_next++] = c;
    if (chb_out_next > CHB_BUF_SIZE-1)
        chb_out_next = 0;

    state = disable_ints();
    if ((inp(SCCB_CMD) & 0x04))
    {
        outp(SCCB_DATA, chb_out_buf[chb_out_now++]);

        if (chb_out_now > CHB_BUF_SIZE-1)
            chb_out_now = 0;
    }
    if (state)
        enable_ints();

}  /* End of serial_out() */


/******************************************************************************
 *
 *  scc_isr()
 *
 ******************************************************************************/

void    interrupt far scc_isr(
                            unsigned es, unsigned ds, unsigned di, unsigned si,
                            unsigned bp, unsigned sp, unsigned bx, unsigned dx,
                            unsigned cx, unsigned ax, unsigned ip, unsigned cs,
                            unsigned flags)
{
    static WORD  save_rr0b = 0;
    static WORD  save_rr0a = 0;
    static WORD  save_rr1b = 0;
    static WORD  rr2b, rr3a;
    static WORD  rr0b, rr1b;
    static WORD  rr0a, rr1a;


    jim++;


    do
    {
        /* First, read the Interrupt Vector from Read Register 2.  This is a
         * unique SCC register (i.e. it is not duplicate for both channels.
         * Furthermore, this register is accessed from channel B.
         */
        inp(SCCB_CMD);          /* Reset pointer bits */
        outp(SCCB_CMD, 2);
        rr2b = inp(SCCB_CMD);
        switch(rr2b)
        {
            case 0x00:   /* Ch B Tx buffer empty */
                b1++;
                if (chb_out_now != chb_out_next)
                {
                    outp(SCCB_DATA, chb_out_buf[chb_out_now++]);
                    if (chb_out_now > CHB_BUF_SIZE-1)
                        chb_out_now = 0; /* time to wrap-around */
                }

                else/* no more to send out */
                {
                    outp(SCCB_CMD, 0x28); /* WR0B = 0x28 */
                }

                outp(SCCB_CMD, 0x38);     /* reset highest IUS */

                break;


            case 0x02:   /* Ch B Ext/Status change */
                b2++;
                outp(SCCB_CMD, 0);
                rr0b = inp(SCCB_CMD);

                if ((save_rr0b & 0x80) ^ (rr0b & 0x80))
                    /* change in break/abort status */
                    inp(SCCB_DATA);
```

214

```c
        if (rr0b & 0x40)  /* Tx underrun/EOM detected */
        {
            outp(SCCB_CMD, 0xC0);
        }

        if (rr0b & 0x04)  /* Tx buffer empty */
        {
            if (chb_out_now != chb_out_next)
            {
                outp(SCCB_DATA, chb_out_buf[chb_out_now++]);
                if (chb_out_now > CHB_BUF_SIZE-1)
                    chb_out_now = 0;  /* time to wrap-around */
            }

            else/* no more to send out */
            {
                outp(SCCB_CMD, 0x28);  /* WR0B = 0x28 */
            }
        }

        if (rr0b & 0x01)  /* Rx data available */
        {
            chb_in_buf[chb_in_next++] = inp(SCCB_DATA);
            if (chb_in_next > CHB_BUF_SIZE-1)
                chb_in_next = 0;       /* time to wrap-around */
        }

        save_rr0b = rr0b;

        outp(SCCB_CMD, 0x10);     /* reset ext/status interrupt */
        outp(SCCB_CMD, 0x30);     /* reset special Rx  cond. status */
        outp(SCCB_CMD, 0x38);     /* reset highest IUS */
        break;


case 0x04:   /* Ch B Rx data ready */
    b3++;
    chb_in_buf[chb_in_next++] = inp(SCCB_DATA);
    if (chb_in_next > CHB_BUF_SIZE-1)
        chb_in_next = 0;       /* time to wrap-around */

    outp(SCCB_CMD, 0x30);     /* reset special Rx  cond. status */
    outp(SCCB_CMD, 0x38);     /* reset highest IUS */
    break;


case 0x06:   /* Ch B Special Rx condition */
    b4++;
    outp(SCCB_CMD, 0x01);
    rr1b = inp(SCCB_CMD);

    if (rr1b & 0x20)
    {
        /* Rx overrun error */
    }

    if (rr1b & 0x01)
    {
        /* data has cleared the (SCC) transmitter */
    }

    outp(SCCB_CMD, 0x30);     /* reset special Rx  cond. status */
    outp(SCCB_CMD, 0x38);     /* reset highest IUS */
    break;


case 0x08:   /* Ch A Tx buffer empty */
    a1++;
    break;


case 0x0A:   /* Ch A Ext/Status change */
    a2++;
    outp(SCCA_CMD, 0x00);
    rr0a = inp(SCCA_CMD);

    if ((save_rr0a & 0x80) ^ (rr0a & 0x80))
    /* change in break/abort status */
    {
        a_brk_abort++;
        /* Force PA-100 into reacquire */
    }

    if (rr0a & 0x40)       /* Tx underrun/EOM */
```

```c
            a_txunderrun_eom++;
            /* outp(SCCA_CMD, 0xC0); */

        if (rr0a & 0x10)
        {
            hunt = TRUE;
            /* ignore break/abort */
            /* setup DMA for frame receive */
        }
        else
        {
            hunt = FALSE;
            /* allow break/abort */
        }

        if (rr0a & 0x04) /* Tx buffer empty */
            a1++;

        if (rr0a & 0x01) /* Rx character available */
            a3++;

        save_rr0a = rr0a;

        outp(SCCA_CMD, 0x10);    /* reset ext/status interrupt */
        outp(SCCA_CMD, 0x30);    /* reset special Rx cond. status */
        outp(SCCA_CMD, 0x38);    /* reset highest IUS */
        break;


    case 0x0C:   /* Ch A Rx data ready */
        a3++;
        break;


    case 0x0E:   /* Ch A Special Rx condition */
        a4++;
        outp(SCCA_CMD, 0x01);
        rr1a = inp(SCCA_CMD);

        if (rr1a & 0x20) /* Rx overrun error */
        {
            a_rxoverrun++;
        }

        if ((rr1a & 0xC0) == 0x80)         /* EOF with NO CRC error */
        {
            outpw(D0CON, 0xA3A4); /* STOP DMA 0 - no more Rx */
            cha_in_buf0[514 - inpw(D0TC) - 2] = NULL_CHAR;
            /* make it a char string */
            rx_eom = TRUE;

            /* ignore break/abort */
        }

        if ((rr1a & 0xC0) == 0xC0)         /* EOF with CRC error */
        {
            outpw(D0CON, 0xA3A4);     /* STOP DMA 0 - no more Rx */
            cha_in_buf0[0] = NULL_CHAR;   /* make it a char string */
            rx_eom = FALSE;

            /* ignore break/abort */
        }

        outp(SCCA_CMD, 0x30);
        outp(SCCA_CMD, 0x38);
        break;

    default:
        /* Error */
        break;

} /* End of SWITCH(rr2b) */


/* Check for any pending ints, which will cause the entire SWITCH above
 * to reiterate.
 */
inp(SCCA_CMD);
outp(SCCA_CMD, 3);
rr3a = inp(SCCA_CMD);

} while (rr3a != 0);
```

```c
        /* Send non-specific EOI to Interrupt Controller */
        outpw(0xFF22, 0x8000);

}   /* End of scc_isr() */



/***************************************************************************
 *
 *   scca_wreg()
 *
 ***************************************************************************/

void scca_wreg(int reg, int value)
{
    if (debug)
        dprint("SCCA WR: %X = %X", reg, value);

    outp(SCCA_CMD, reg);
    outp(SCCA_CMD, value);

}   /* End of scca_wreg() */



/***************************************************************************
 *
 *   sccb_wreg()
 *
 ***************************************************************************/

void sccb_wreg(int reg, int value)
{
    if (debug)
        dprint("SCCB WR: %X = %X", reg, value);

    outp(SCCB_CMD, reg);
    outp(SCCB_CMD, value);

}   /* End of sccb_wreg() */



/***************************************************************************
 *
 *   scc_hunt)
 *
 ***************************************************************************/

void scc_hunt(void)
{
    unsigned long int i;


    dprint("Entering Hunt Mode");

    outp(SCCA_CMD, 3);
    outp(SCCA_CMD, 0xD8);      /* Disable receiver */
    outp(SCCA_CMD, 3);
    outp(SCCA_CMD, 0xD9);      /* Enable receiver & enter Hunt mode */

    for (i = 0; i < 0x0001FFFFL; i++)
    {
        if ((i%0x3FFFL) == 0)
            dprint(".");

        if (!(inp(SCCA_CMD) & 0x010))
            break;
    }

    if (inp(SCCA_CMD) & 0x010)
        dprint("Sync (Flag) not detected, still in Hunt Mode\n");
    else
        dprint("Sync (Flag) detected!\n");

}   /* End of scc_hunt() */
```

End of scc.h, scc.c

scenario.h, scenario.h

```
/*************************************************************************
 *
 *   SCENARIO.H
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *
 *************************************************************************/


/*************************************************************************
 *
 *   SCENARIO.C
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+---------------------------------------------
 *
 *************************************************************************/

#include     "gen_defs.h"

#define       SCENARIO
#include "cmd.h"
#undef        SCENARIO
```

End of scenario.h, scenario.c

spacket.h, spacket.c

```
/*****************************************************************************
 *
 *   SPACKET.H
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+-----------------------------------------------
 *
 *****************************************************************************/


/*****************************************************************************
 *
 *   SPACKET.C
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+-----------------------------------------------
 *
 *****************************************************************************/

#include     "gen_defs.h"

#define      SPACKET
#include "spacket.h"
#undef        SPACKET
```

End of spacket.h, spacket.c

219

startup.asm

```
;**************************************************************************
;*
;*  80186 C Startup
;*
;*  This is the beginning of ROM code for PANSAT.
;*
;*
;*  Revision History
;*
;*  When                    Who             What
;*  ---------------+-------+------------------------------------------
;*  28 March 1996        Jah                Adopt for Pansat DCS
;*
;**************************************************************************


FALSE       equ    0
TRUE        equ    1


;**************************************************************************
;*
;*  MACRO: transmit
;*
;**************************************************************************

transmit    MACRO
    LOCAL   transmit_wait

    mov     bx, ax

transmit_wait:
    in      al, SCCB_CMD
    and     al, 4
    jz      transmit_wait

    mov     ax, bx
    out     SCCB_DATA, al

ENDM


;**************************************************************************
;*
;*  MACRO: DELAY
;*
;**************************************************************************

delay   MACRO   delay_time
    LOCAL   delay_loop

    mov     cx, delay_time

delay_loop:
    dec     cx
    jnz     delay_loop

ENDM


;**************************************************************************
;*
;*  MACRO: DISPLAY
;*
;**************************************************************************

display     MACRO
    LOCAL   display1
    LOCAL   display2

    shr     ax, cl
    and     al, 0Fh
    cmp     al, 9
    jb      display1
    add     al, 037h            ;adjust A-F
    jmp     display2
display1:
    add     al, '0'             ;adjust 0-9
display2:
    transmit
```

ENDM


```
; **************************************************************************
; Specify stack size.
; **************************************************************************

STACK_SIZE      EQU     800H        ;defined in WORDS (2 bytes) = 4k bytes


; **************************************************************************
; Place EXTRN statements for interrupt handling routines outside
; of all SEGMENT/ENDS pairs.
; **************************************************************************
; For example, suppose you have an interrupt handling routine written in C
; with the function name int_hdlr().
; In order for you to reference the routine in this file, you should declare
; _int_hdlr as an external reference (note that the C compiler prefixes an
; underscore character to the function name).  The declaration should be
; placed outside of all SEGMENT/ENDS pairs as follows:
;       EXTRN _int_hdlr:FAR
; Refer to the sample code below that initializes the interrupt vector table.
;


; **************************************************************************
; The primary function of the start-up code is to set up the run-time
; environment before passing control to C function main().
; The start-up code performs the following functions:
; 1) Initialize hardware and check RAM.
; 2) Copy initializers from ROM to RAM to setup initialized program variables
;    to proper initial values.
; 3) Zero all uninitialized program variables.
; 4) Initialize interrupt vector table, if necessary.
; 5) Setup data segment.
; 6) Setup stack segment.
; 7) Pass control to C function main().


PUBLIC   __acrtused
__acrtused      EQU     1


; The symbol __acrtused has to be in lower-case.  Starting from version 4,
; when the Microsoft C optimizing compiler compiles a C file, it places an
; external reference to this symbol in the object file.  The public definition
; of __acrtused is contained in an object module called crt0.  This module is
; placed in the Microsoft C run-time library file.  This module contains the
; start-up code for the DOS environment.  So when you link your C object files
; with the run-time library file, the linker will extract the start-up object
; module from the run-time library file to satisfy the external references.
; As a result, the start-up code for DOS environment will be included in the
; linker output.
; If you do not make use of any function supplied in the run-time library
; file, you do not link with the library file at all.  Then you have to
; include a public definition of __acrtused in this file to resolve all the
; external references in the C object files.
; Even though you are building an application for an embedded environment,
; sometimes you may want to link with the run-time library file.  The reason
; is that the run-time library file contains both DOS-dependent functions,
; such as printf(), and DOS-independent functions, such as strcpy().  If you
; want to make use of certain DOS-independent functions that are contained in
; the run-time library file, you would link with the library file.  If the
; link map shows that the linked module contains the crt0 module, you know
; that you have linked in some DOS-dependent functions from the run-time
; library file.
; Make sure you specify this start-up file as the first object file
; in the linker command line, then the other object files and place the
; modified combined run-time library file as the last file.
;
; The following memory map shows a typical run-time environment of an embedded
; application developed using the Microsoft C optimizing compiler.
; It will help you understand the start-up code presented in this file.
; The naming convention of class names presented here conforms with the
; Microsoft C optimizing compiler, version 4 and up.  The names enclosed in
; single quotes are class names of segments.
;
;  High address
;  -            ----------------
;  |           |                |
;  | FFFF:0 ---------------- <-- Bootstrap code (JMP FAR PTR START_)
;  |           :                :
;  |           ----------------  <--
;  |          | [FAR_DATA]    |  ^  This area of ROM contains initializers that
```

```
;   |            ---------------  |    are used by start-up routine to initialize
;   |            | [CONST]      | | |  segments with these class names in RAM at
;   |            ---------------  | |  power-up.
;   |            | [DATA]       | | |  (The INITDATA control causes the PROM86
;   R            ---------------  | |  utility to place initializers between address
;   O            | [DATA_BEG]   | |  v  labels _bdata and _edata and between _bfdata
;   M            --------------- <-- and _efdata in this area of ROM)
;   |            | 'CODE_END'   | <---- This class contains only one segment of zero
;   | _etext ---------------         length. Initializers are located at _etext.
;   |            | 'CODE'       | <-- Text segments with class name CODE.
;   | _start_---------------
; - START_ :                :
;                           :                :
; -                         :                :
;   |            --------------- <----- End of DGROUP (size of DGROUP <= 64K bytes)
;   |            | 'STACK'      | <-- This class contains the stack.
;   |            ---------------
;   |            | 'BSS_END'    |
;   | _end       ---------------
;   |            | 'BSS'        | <-- This class contains uninitialized data.
;   |            ---------------
;   |            | 'DATA_END'   |
;   | _edata --------------- <----
;   |            | 'CONST'      |  ^ These three classes of segments are to be
;   R            ---------------  |  initialized with initializers in ROM
;   A            | 'DATA'       | |  by start-up routine at power-up.
;   M            ---------------  |
;   |            | 'DATA_BEG'   |  v
;   | _bdata --------------- <---- Start of DGROUP
;   |            |'HUGE_BSS_END'|
;   | _ehbss ---------------
;   |            | 'HUGE_BSS'   | <-- This class contains uninitialized data.
;   |            ---------------
;   |            |'HUGE_BSS_BEG'|
;   | _bhbss ---------------
;   |            | 'FAR_BSS_END'|
;   | _efbss ---------------
;   |            | 'FAR_BSS'    | <-- This class contains uninitialized data.
;   |            ---------------
;   |            | 'FAR_BSS_BEG'|
;   | _bfbss ---------------
;   |            |'FAR_DATA_END'|
;   | _efdata---------------
;   |            | 'FAR_DATA'   | <-- This class contains initialized data.
;   |            ---------------
;   |            |'FAR_DATA_BEG'|
;   | _bfdata---------------
;   |            :                :
;   |            ---------------
;   |            |              | | <-- The interrupt vector table should be
; -     0:0 ---------------         initialized by start-up routine at power-up.
;   Low address
;
; NOTE:
;       It is important to maintain the order of SEGMENTS/ENDS pairs as shown
;       below.
;
;       Text segments all have class name CODE.  They contain program code
;       which is machine instructions generated by the compiler.
;
;       Data segments with class names DATA_BEG, DATA, CONST, BSS and STACK
;       belong to a group named DGROUP.  Since these segments belong to
;       a group, it follows that the total memory space occupied by them
;       cannot exceed 64K bytes.
;
;       The object files may contain data segments with class names
;       FAR_DATA, FAR_BSS and HUGE_BSS.
;       These classes of segments are generated by the Microsoft C
;       optimizing compiler to support 'far' and 'huge' data objects.
;       Check your C manual for details.
;
;       Data segments with class names FAR_BSS and HUGE_BSS contain
;       'far' and 'huge' uninitialized data, respectively. These segments
;       do not belong to any group.  They should be located before
;       the group DGROUP in the RAM space of your target system.  The
;       advantage of locating these segments before DGROUP is that you may
;       use the memory space from the end of STACK segment to the end of
;       RAM as heap to implement your own memory allocation scheme.
;
;       For 'huge' data objects, multiple segments may be generated by the
;       C compiler to hold a single data object that is greater than 64K
;       bytes in size.  These segments must be located together in proper
;       order.
;
```

222

```
;       Data segments with class name FAR_DATA contain 'far' and 'huge'
;       initialized data.  They should also be located before the group
;       DGROUP in the RAM space of your target system.
;
;       When you prepare data file for programming eprom, the initializers
;       for segments with class names DATA_BEG, DATA, CONST and FAR_DATA
;       have to be placed in ROM addresses behind the CODE_END class.
;       The INITDATA control of PROM86, version 5.2 and up, performs this
;       operation for you.  At power-up, the start-up routine will copy these
;       initializers from ROM to initialize the appropriate variables in RAM.
;
;       In order to make use of the INITDATA control in PROM86, you
;       must preserve the following public labels in this start-up file:
;       _bfdata - beginning of initializers in FAR_DATA class
;       _efdata - end of initializers in FAR_DATA class
;       _bdata  - beginning of initializers in DGROUP group
;       _edata  - end of initializers in DGROUP group
;       _etext  - end of code
;
;       If you want PROM86 to determine the extraction address range,
;       i.e. the ADDRESSES option is not specified in PROM86 v6.0, you must
;       preserve the public label _start_ in this start-up file.
;
;       In addition to the above labels, the routine in this start-up file
;       uses the following public labels:
;       _end    - end of uninitialized data in BSS class
;       _bfbss  - beginning of uninitialized data in FAR_BSS class
;       _efbss  - end of uninitialized data in FAR_BSS class
;       _bhbss  - beginning of uninitialized data in HUGE_BSS class
;       _ehbss  - end of uninitialized data in HUGE_BSS class


;*****************************************************************************
;*****************************************************************************
;*
;*  Segment declarations
;*
;*****************************************************************************
;*****************************************************************************


;*****************************************************************************
;*
;*  BEGFDATA
;*
;*****************************************************************************

BEGFDATA SEGMENT PARA PUBLIC 'FAR_DATA_BEG'

PUBLIC  _bfdata
_bfdata LABEL BYTE       ; This label marks the beginning of initialized data
             ; in FAR_DATA class.
BEGFDATA ENDS


;*****************************************************************************
;*
;*  FAR_DATA_START
;*
;*****************************************************************************

FAR_DATA_START SEGMENT PARA PUBLIC 'FAR_DATA'
FAR_DATA_START ENDS

; By default, the locator places segments with the
; same class name together.
; The purpose of the FAR_DATA_START segment is to cause the locator
; to locate all segments with class name FAR_DATA that contain
; initialized variables between the BEGFDATA and ENDFDATA segments.

EMULATOR_DATA  segment para public 'FAR_DATA'
; Segment contains data for the floating point emulator.
EMULATOR_DATA  ends

;*****************************************************************************
;*
;*  ENDFDATA
;*
;*****************************************************************************

ENDFDATA SEGMENT PARA PUBLIC 'FAR_DATA_END'

PUBLIC  _efdata
_efdata LABEL BYTE       ; This label marks the end of initialized data
```

```
                  ; in FAR_DATA class.
ENDFDATA ENDS


;*****************************************************************************
;*
;*   BEGFBSS
;*
;*****************************************************************************

BEGFBSS SEGMENT PARA PUBLIC 'FAR_BSS_BEG'

PUBLIC  _bfbss
_bfbss  LABEL BYTE       ; This label marks the beginning of uninitialized
                  ; data in FAR_BSS class.
BEGFBSS ENDS


;*****************************************************************************
;*
;*   FAR_BSS_START
;*
;*****************************************************************************

FAR_BSS_START SEGMENT PARA PUBLIC 'FAR_BSS'
FAR_BSS_START ENDS


; By default, the locator places segments with the same class name
; together.
; The purpose of the FAR_BSS_START segment is to cause the locator to locate
; all segments with class name FAR_BSS between the BEGFBSS and ENDFBSS
; segments.
; Segments with class name FAR_BSS contain uninitialized variables.


;*****************************************************************************
;*
;*   ENDFBSS
;*
;*****************************************************************************

ENDFBSS SEGMENT PARA PUBLIC 'FAR_BSS_END'

PUBLIC  _efbss
_efbss  LABEL BYTE       ; This label marks the end of uninitialized data
                  ; in FAR_BSS class.
ENDFBSS ENDS


;*****************************************************************************
;*
;*   BEGHBSS
;*
;*****************************************************************************

BEGHBSS SEGMENT PARA PUBLIC 'HUGE_BSS_BEG'

PUBLIC  _bhbss
_bhbss  LABEL BYTE       ; This label marks the beginning of uninitialized
                  ; data in HUGE_BSS class.
BEGHBSS ENDS


;*****************************************************************************
;*
;*   HUGE_BSS_START
;*
;*****************************************************************************

HUGE_BSS_START SEGMENT PARA PUBLIC 'HUGE_BSS'
HUGE_BSS_START ENDS


; By default, the locator places segments with the same class name
; together.
; The purpose of the HUGE_BSS_START segment is to cause the locator to locate
; all segments with class name HUGE_BSS between the BEGHBSS and ENDHBSS
; segments.
; Segments with class name HUGE_BSS contain uninitialized variables.


;*****************************************************************************
;*
```

```
;*    ENDHBSS
;*
;*****************************************************************************

ENDHBSS SEGMENT PARA PUBLIC 'HUGE_BSS_END'

PUBLIC  _ehbss
_ehbss  LABEL BYTE       ; This label marks the end of uninitialized data
                 ; in HUGE_BSS class.
ENDHBSS ENDS



;*****************************************************************************
;*
;*    DGROUP segments
;*
;*****************************************************************************

; DGROUP GROUP NULL, _DATA, CONST, ENDDATA, _BSS, ENDBSS, STACK

DGROUP GROUP NULL,_DATA,PSP,CDATA,CONST,HDR,MSG,PAD,EPAD,ENDDATA,_BSS,ENDBSS,STACK

;*****************************************************************************
;*
;*    NULL
;*
;*    This segment contains 8 bytes of zeros.  If a (DS:0) null pointer assignment
;*    occurs, these byte locations will be overwritten.  You may implement your
;*    own routine to check for null pointer assignment.
;*
;*****************************************************************************

NULL SEGMENT PARA PUBLIC 'DATA_BEG'

PUBLIC  _bdata           ; This label marks the beginning of initialized data.
_bdata  LABEL BYTE
    DB 8 DUP (0)

NULL ENDS



;*****************************************************************************
;*
;*    _DATA
;*
;*    Segment with class name DATA contains initialized variables.
;*
;*****************************************************************************

_DATA SEGMENT WORD PUBLIC 'DATA'
; segment contains initialized variables
    PUBLIC  __fac
__fac   DQ      0        ;FP Accumulator
    PUBLIC  _errno
_errno DW      0        ;Initial Error Code
_DATA ENDS


PSP SEGMENT PARA PUBLIC 'DATA' ; MUST BE PARAGRAPH ALIGNED
; Segment contains data for initializing floating point emulator.
PSP ENDS

CDATA SEGMENT WORD COMMON 'DATA'
        DW      0  ; DO NOT DEFINE ANY VARIABLE IN THIS SEGMENT
__fpinit LABEL DWORD
    PUBLIC __fpinit
CDATA ENDS



;*****************************************************************************
;*
;*    CONST
;*
;*    Segment with class name CONST contains constants.
;*
;*****************************************************************************

CONST SEGMENT WORD PUBLIC 'CONST'
CONST ENDS


HDR SEGMENT BYTE PUBLIC 'MSG'    ; HEADER SEGMENT OF ERROR MESSAGE STRINGS
    DB '<<NMSG>>'
HDR ENDS
```

```
MSG SEGMENT BYTE PUBLIC 'MSG'    ; ERROR MESSAGE STRINGS
MSG ENDS


PAD SEGMENT BYTE COMMON 'MSG'    ; ERROR MESSAGE PADDING MARKER
        DW -1
PAD ENDS


EPAD SEGMENT BYTE COMMON 'MSG'   ; END OF PADDING MARKER
        DB -1
EPAD ENDS



;*****************************************************************************
;*
;*   ENDDATA
;*
;*****************************************************************************

ENDDATA SEGMENT PARA PUBLIC 'DATA_END'

PUBLIC  _edata
_edata  LABEL BYTE       ; This label marks the end of initialized data.

ENDDATA ENDS



;*****************************************************************************
;*
;*   _BSS
;*
;*   Segment with class name BSS contains uninitialized variables.
;*
;*****************************************************************************

_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS



;*****************************************************************************
;*
;*   ENDBSS
;*
;*****************************************************************************

ENDBSS  SEGMENT WORD PUBLIC 'BSS_END'

PUBLIC  _end
_end    LABEL BYTE       ; This label marks the end of uninitialized data.
ENDBSS  ENDS



;*****************************************************************************
;*
;*   STACK
;*
;*****************************************************************************

STACK   SEGMENT PARA STACK 'STACK'

    DW STACK_SIZE DUP (?)

stack_top LABEL WORD

STACK   ENDS



;*****************************************************************************
;*
;*   _TEXT
;*
;*****************************************************************************

_TEXT SEGMENT PARA PUBLIC 'CODE'
EXTRN   _main:NEAR       ;C main()

    ASSUME  CS:_TEXT
    ASSUME  DS:DGROUP, SS:DGROUP


;*      80186 Initialization

    UMCS        equ     0FFA0h      ;upper memory chip select
    UMCS_DATA   equ     0F038h      ;start of EPROM F000:0, 64K
```

```
                                        ;  external Ready ignored, 0 wait states

;*      Lower Memory Chip Selects are NOT used

    PACS      equ      0FFA4h
    PACS_DATA    equ      0000h        ;PCS0 (base) = 0, bus ready must be active
                                       ;  to complete a bus cycle, no wait states
                                       ;  inserted in bus cycle

    MMCS      equ      0FFA6h
    MMCS_DATA    equ      00000h       ;Start = 0:0, 512K in size

    MPCS      equ      0FFA8h
    MPCS_DATA    equ      04087h       ;512K block, + PCS5/6
                                       ;  PCSx go active for i/o bus cycles,
                                       ;  requires bus ready be active to complete
                                       ;  bus cycle (applies to PCS4-PCS6),
                                       ;  0 wait states inserted for PCS4-PCS6

PUBLIC START_
START_:
PUBLIC _start_              ;Must be paragraph aligned (i.e. offset is 0)
_start_:                    ;  and the address where program code starts.

;*      80186 Initialization

    mov      dx, MMCS       ,
    mov      ax, MMCS_DATA
    out      dx, ax

    mov      dx, MPCS
    mov      ax, MPCS_DATA
    out      dx, ax

    mov      dx, PACS
    mov      ax, PACS_DATA
    out      dx, ax

;* For the test board w/ separate 1.8MHz SCC clock
;    mov dx, 0FFA2h
;    mov ax, 03FF8h
;    out dx, ax
;    mov dx, 0FFA4h
;    mov ax, 0037h
;    out dx, ax
;    mov dx, 0FFA6h
;    mov ax, 041F8h
;    out dx, ax
;    mov dx, 0FFA8h
;    mov ax, 0A038h
;    out dx, ax


;****************************************************************************
;*
;*  Init Peripheral Control Bus
;*  8255 (PPI) at 0xC0, 0xC2, 0xC4, 0xC6
;*  Mode 2: Port A = bidirectional using Port C as handshaking
;*          Port B = Address selections and Read & Write strobes
;*          Port C = Handshaking lines
;*
;*  After PPI init, Port B is set to 0xC0 which sets the Read and Write
;*  lines high, which clears them since they are active low signals.
;*
;****************************************************************************

PCB_PPI_BASE    equ         0100h
PCB_PPI_PortA   equ         PCB_PPI_BASE + 0
PCB_PPI_PortB   equ         PCB_PPI_BASE + 2
PCB_PPI_PortC   equ         PCB_PPI_BASE + 4
PCB_PPI_Ctrl    equ         PCB_PPI_BASE + 6

; Port C single bit set/reset
PCB_PortC_Set0       equ    01h         ;Sets Port C, bit 0 to 1
PCB_PortC_Reset0     equ    00h         ;Sets Port C, bit 0 to 0
PCB_PortC_Set1       equ    03h         ;Sets Port C, bit 1 to 1
PCB_PortC_Reset1     equ    02h         ;Sets Port C, bit 1 to 0
PCB_PortC_Set2       equ    05h         ;Sets Port C, bit 2 to 1
PCB_PortC_Reset2     equ    04h         ;Sets Port C, bit 2 to 0


    mov      dx, PCB_PPI_Ctrl
    mov      al, 0C0h
    out      dx, al                      ;Setup via the Control port
```

```
        mov     dx, PCB_PPI_PortB
        out     dx, al                      ;Output to port b (Read & Write HIGH)

        mov     dx, PCB_PPI_PortA
        sub     al, al
        out     dx, al                      ;Output to port a (Data = all zeros)

        mov     dx, PCB_PPI_Ctrl
        mov     al, PCB_PortC_Set1          ;PC1 = PPI Input Strobe
        out     dx, al
        mov     al, PCB_PortC_Set0          ;Modem Power (Active LOW -> keep it OFF)
        out     dx, al


; Clear power settings on EPS (Ports 0 and 2)
        mov     dx, PCB_PPI_PortA
        mov     al, 0
        out     dx, al

        mov     dx, PCB_PPI_PortB
        mov     al, 0E8h            ;11 10 1000
        out     dx, al

        mov     al, 0A8h            ;10 10 1000
        out     dx, al

        mov     al, 0E8h            ;11 10 1000
        out     dx, al

        mov     dx, PCB_PPI_PortB
        mov     al, 0C8h            ;11 00 1000
        out     dx, al

        mov     al, 088h            ;10 00 1000
        out     dx, al

        mov     al, 0C8h            ;11 00 1000
        out     dx, al


;*****************************************************************************
;*
;*  Init 85C30 (Channel B) for:
;*              asynchronous, 9600 bps, no parity, 1 stop bit, 8 bits/char
;*
;*****************************************************************************

        SCCB_CMD        equ     0                       ;Channel B Command
        SCCB_DATA       equ     4                       ;Channel B Data
        SCCA_CMD        equ     2                       ;Channel A Command
        SCCA_DATA       equ     6                       ;Channel A Data

        BTABLE_LEN      equ (OFFSET sccb_table_end) - (OFFSET sccb_table)
        ATABLE_LEN      equ (OFFSET scca_table_end) - (OFFSET scca_table)

        IOCON           equ     0FF38h
        IOCON_INIT      equ     00Ch                    ;Edge-triggered, turned OFF, priority 4
        INT0_ISR    equ     12


;public init_scca
;init_scca:
        in              al, SCCA_CMD    ;read status

        mov             bx, ATABLE_LEN
        mov             bp, OFFSET scca_table

public table_loada
table_loada:
        mov             al, cs:[bp]
        out             SCCA_CMD, al
        inc             bp
        dec             bx
        jnz             table_loada

init_sccb:
        in              al, SCCB_CMD    ;read status

        mov             bx, BTABLE_LEN
        mov             bp, OFFSET sccb_table

public table_loadb
table_loadb:
```

228

```asm
        mov             al, cs:[bp]
        out             SCCB_CMD, al
        inc             bp
        dec             bx

        jnz             table_loadb

public table_loadb_done
table_loadb_done:
        mov     al, '*'
        transmit

        jmp             edac_done
        jmp             init_edac

scca_table:
        db      009h                    ;point to WR9
        db      0C0h                    ;force hardware reset
        ;
        db      004h                    ;point to WR4
        db      020h                    ;x1 clock, SDLC mode, SYNC mode
        ;
        db      001h                    ;point to WR1
        db      000h                    ;disable DMA and interrupts
        ;
        db      002h                    ;point to WR2
        db      000h                    ;zero interrupt vectors
        ;
        db      003h                    ;point to WR3
        db      0C8h                    ;Rx 8 bits, Rx CRC enabled
        ;
        db      005h                    ;point to WR5
        db      061h                    ;Tx 8 bits, Tx CRC enabled
        ;
        db      006h                    ;point to WR6
        db      000h                    ;SDLC address field
        ;
        db      007h
        db      07Eh
        ;
        db      00Fh                    ;point to WR15
        db      091h                    ;Enable Int: Break/Abort, Sync/Hunt
        ;
        db      007h                    ;point to WR7
        db      063h                    ;Extended Read, get CRC bits, AUTO RTS, EOM, TX flag
        ;
        db      009h                    ;point to WR9
        db      022h                    ;No vector and no INTACK
        ;
        db      00Ah                    ;point to WR10
        db      0A0h                    ;CRC preset=1, NRZI, Flag idle
        ;
        db      00Bh                    ;point to WR11
        db      009h                    ;TxCK=TRxC, RxCK=RTxC
;;      db      056h                    ;TxCLK=BRG, RxCLK=BRG, TRxC=output using BRG
        ;
;       db      00Ch                    ;point to WR12
;       db      45                          ;TCL: to give 78.125 KHz clock
        ;
;       db      00Dh                    ;point to WR13
;       db      000h                    ;TCH
        ;
        db      00Eh                    ;point to WR14
        db      000h                    ;BRG=RTxC pin
;       db      002h                        ;BRG=PCLK
        ;
        db      00Eh                    ;point to WR14
        db      005h                    ;Enable BRG, BRG=RTxC pin, DTR/Request Function
;       db      007h                        ;Enable BRG, BRG=PCLK, DTR/Request function
        ;
        db      003h                    ;point to WR3
        db      0D9h                    ;Enable RX, CRC, enter hunt
        ;
        db      005h                    ;point to WR5
        db      069h                    ;Enable TX, CRC
        ;
        db      000h                    ;point to WR0
        db      080h                    ;Reset TxCRC
        ;
        db      000h                    ;point to WR0
        db      010h                    ;reset ext/status
        ;
        db      000h                    ;point to WR0
        db      010h                    ;reset ext/status
```

```
            ;
            db      001h                        ;point to WR1
;           db      0F9h                         ;DMA Request Mode, Int on Special Rx, EXT INT enable
            db      0F8h                     ;DMA Request Mode, Int on Special Rx
            ;
            db      009h                        ;point to WR9
            db      02Ah                        ;enable interrupts

scca_table_end:


sccb_table:
; based on SCCB using PCLK at 7.3728 MHz, x16 clock mode, at 38400 b/s
      db        04h, 044h         ;WR4, x16 clock, 8 bit sync, 1 stop bit, no parity

      db        01h, 012h         ;WR1, INT on all Rx or special, Int on Tx empty

      db        03h, 0C0h         ;WR3, Rx: 8 bits/char, Rx disable
      db        05h, 060h         ;WR5, Tx: 8 bits/char, Tx disable

      db        0Bh, 056h         ;WR11, Rx: BRG, Tx: BRG, TRxC: BRG
;     db        0Ch, 016h           ;WR12, lower TC        ; this is for 9600 b/s
      db        0Ch, 0Ah        ;WR12, lower TC       ; this is for 19.2 kb/s
;     db        0Ch, 04h        ;WR12, lower TC       ; this is for 38.4 kb/s

      db        0Dh, 000h         ;WR13, upper TC
      db        0Eh, 003h       ;WR14, BRG use PCLK, set DTR, enable BRG

      db        03h, 0C1h         ;WR3: ... + Rx enable
      db        05h, 06Ah         ;WR5: ... + Tx enable, /RTS

      db        00h, 010h         ;WR0: Reset EXT/Status Interrupts
      db        00h, 028h         ;WR0: Reset TxINT Pending

      db        09h, 02Ah         ;WR9: Enable Interrupts


; based on SCCB using BRG, with 1.8432 MHz clock, with x16 clock mode, @ 9600 bps
;     db    04h, 044h                 ;WR4, x16 clock, 8 bit sync, 1 stop bit, no parity

;     db    01h, 12h              ;WR1, INT on all Rx or special, Int on Tx empty

;     db    03h, 0C0h                 ;WR3, Rx: 8 bits/char, Rx disable
;     db    05h, 060h                 ;WR5, Tx: 8 bits/char, Tx disable

;     db    0Bh, 056h                 ;WR11, Rx: BRG, Tx: BRG, TRxC: BRG
;     db    0Ch, 004h                 ;WR12, lower TC
;     db    0Dh, 000h                 ;WR13, upper TC
;     db    0Eh, 001h                 ;WR14, set DTR and enable baud gen

;     db    03h, 0C1h                 ;WR3: ... + Rx enable
;     db    05h, 068h                 ;WR5: ... + Tx enable

;     db    00h, 010h                 ;WR0: Reset EXT/Status Interrupts
;     db    00h, 028h                 ;WR0: Reset TxINT Pending

;     db    09h, 02Ah                 ;WR9: Enable Interrupts
sccb_table_end:


;****************************************************************************
;*
;*  Init EDAC
;*
;*      INT2 = Hard Error
;*      INT3 = Soft Error
;*
;****************************************************************************

EXTRN    _edac_soft_isr:NEAR        ;C routine
EXTRN    _edac_hard_isr:NEAR        ;C routine

I2CON           equ     0FF3Ch
I2CON_INIT      equ     01Eh                    ;Level-triggered, turned OFF, priority 6
I3CON           equ     0FF3Eh
I3CON_INIT      equ     01Fh                    ;Level-triggered, turned OFF, priority 7
IMASK           equ     0FF28h
INT2_ISR        equ     14
INT3_ISR        equ     15

public  init_edac
init_edac:
;EXTRN   _ad_isr:NEAR       ;C ad_isr()

      mov       dx, I3CON
```

```
        mov     ax, I3CON_INIT
        out     dx, ax

        mov     dx, I2CON
        mov     ax, I2CON_INIT
        out     dx, ax


        mov     al, 'E'
        transmit


;Perform the Write/Read Test of 0x55AA
;WRITE
        sub     bx, bx
edac_test1_loopa:
        mov     es, bx
        sub     di, di
        mov     cx, 08000h
        mov     ax, 055AAh
        cld
        rep     stosw
        mov     bx, es
        add     bx, 01000h
        cmp     bx, 08000h
        jb      edac_test1_loopa

;READ-BACK
        sub     bx, bx
edac_test1_loopb:
        mov     es, bx
        sub     di, di
        mov     cx, 08000h
edac_test_loopb1:
        cmp     ax, WORD PTR es:[di]
        jnz     edac_test1_error
        loop edac_test_loopb1
        mov     bx, es
        add     bx, 01000h
        cmp     bx, 08000h
        jb      edac_test1_loopb


;Passed the Write/Read of 0x55AA

;Now perform the Write/Read Test of Mod-257
;WRITE
        sub     bx, bx
edac_test2_loopa:
        mov     es, bx
        sub     di, di
        sub     ax, ax                  ;start w/ 0, count to 257, start over w/ 0
        cld
edac_test2_loopa1:
        mov     BYTE PTR es:[di], al    ;move just a byte
        inc     ax                      ;inc the whole word
        cmp     ax, 258
        jb      edac_test2_skip1
        sub     ax, ax                  ;start over the count
edac_test2_skip1:
        inc     di                      ;next location to fill
        jnz     edac_test2_loopa1;not finished w/ 64k block
        mov     bx, es
        add     bx, 01000h
        cmp     bx, 08000h
        jb      edac_test2_loopa ;keep with same count in AX

;READ-BACK
        sub     bx, bx
edac_test2_loopb:
        mov     es, bx
        sub     di, di
        sub     ax, ax                  ;start w/ 0, count to 257, start over w/ 0
        cld
edac_test2_loopb1:
        cmp     BYTE PTR es:[di], al    ;compare just a byte
        jnz     edac_test2_error
        inc     ax                      ;inc the whole word
        cmp     ax, 258
        jb      edac_test2_skip2
        sub     ax, ax                  ;start over the count
edac_test2_skip2:
        inc     di                      ;next location to fill
        jnz     edac_test2_loopb1;not finished w/ 64k block
```

231

```
             mov      bx, es
             add      bx, 01000h
             cmp      bx, 08000h
             jb       edac_test2_loopb  ;keep with same count in AX
             jmp      edac_vectors

;Passed Write/Read Test of Mod-257


edac_test1_error:
             mov      al, '1'
             jmp      startup_error
edac_test2_error:
             mov      al, '2'
             jmp      startup_error


edac_vectors:
;Setup Interrupt Vectors for EDAC
        sub      ax, ax
        mov      es, ax
        mov      ax, OFFSET _edac_hard_isr
        mov      es:WORD PTR INT2_ISR*4, ax          ;Segment of INT2 vector
        mov      ax, SEG _edac_hard_isr
        mov      es:WORD PTR INT2_ISR*4+2, ax     ;Base of INT2 vector


        sub      ax, ax
        mov      es, ax
        mov      ax, OFFSET _edac_soft_isr
        mov      es:WORD PTR INT3_ISR*4, ax          ;Segment of INT3 vector
        mov      ax, SEG _edac_soft_isr
        mov      es:WORD PTR INT3_ISR*4+2, ax     ;Base of INT3 vector


;Allow the interrupts to occur (to the Interrupt Controller)
;Note: Interrupts are still off to the microprocessor
        mov      dx, IMASK
;       mov      ax, 03Fh                     ;Allow INT2 and INT3 interrupts
                                              ;  CPU 'cli' is still imposed!
        mov      ax, 0FFh                     ;Mask OFF all interrupts
        out      dx, ax

;Reset any EDAC errors
        mov      dx, PCB_PPI_Ctrl
        mov      ax, PCB_PortC_Reset2   ;EDAC Error Acknowledge (Active LOW)
        out      dx, ax
        mov      ax, PCB_PortC_Set2
        out      dx, ax

        mov      al, 'e'
        transmit

edac_done:
;Setup SCC INT0 Interrupt Vector and Configure Controller
;*** DO NOT DO THIS BEFORE THE EDAC TEST OCCURS SINCE THE INTERRUPT VECTORS ARE TRASHED!
EXTRN    _scc_isr:NEAR        ;C scc_isr()

        sub      ax, ax
        mov      es, ax
        mov      ax, OFFSET _scc_isr
        mov      es:WORD PTR INT0_ISR*4, ax              ;Segment of INT0 vector
        mov      ax, SEG _scc_isr
        mov      es:WORD PTR INT0_ISR*4+2, ax     ;Base of INT0 vector
        mov      dx, IOCON
        mov      ax, IOCON_INIT
        out      dx, ax


;****************************************************************************
;*
;*   Init Timer for Clock
;*
;*            Timer 2 Interrupt used, INT 13h (19)
;*
;****************************************************************************

T2CON            equ            0FF66h
T2CNT            equ            0FF60h
T2CMP            equ            0FF62h
T2_ISR           equ            013h


EXTRN    _clock_isr:NEAR                       ;C clock_isr()
public set_timer
```

```
set_timer:
    mov             ax, 0
    mov             es, ax
    mov             ax, OFFSET _clock_isr
    mov             es:WORD PTR 013h*4, ax
    mov             ax, SEG _clock_isr
    mov             es:WORD PTR 013h*4+2, ax        ;Base of TMR2 vector

    mov             dx, T2CNT
    sub             ax, ax
    out             dx, ax                          ;Counter Register

    mov             dx, T2CMP
    mov             ax, 30720               ;Max Count
    out             dx, ax

    mov             dx, T2CON
    mov             ax, 06001h                      ;no inhibit, INT, Continuous
    out             dx, ax
    mov             ax, 0E001h                      ;same as above, but Enable as well
    out             dx, ax


;****************************************************************************
;*
;*  Init A/D Interrupt Service
;*
;*  INT1 = A/D Interrupt
;*
;****************************************************************************

I1CON           equ     0FF3Ah
I1CON_INIT      equ     01Dh                        ;Level-triggered, turned OFF, priority 5
INT1_ISR        equ     13
EXTRN   _ad_isr:NEAR

public set_ad
set_ad:
    mov             ax, 0
    mov             es, ax
    mov             ax, OFFSET _ad_isr
    mov             es:WORD PTR INT1_ISR*4, ax
    mov             ax, SEG _ad_isr
    mov             es:WORD PTR INT1_ISR*4+2, ax        ;Base of AD ISR vector


;****************************************************************************
;*
;*  Perform variable initialization.
;*  Initializers are copied from ROM to RAM.
;*
;****************************************************************************

PUBLIC  _init_begin
_init_begin:

    cld

;Jah
    mov             al, 'I'
    transmit

; Transfer Count
    MOV AX,OFFSET DGROUP:_edata ; Transfer counter
    CMP AX,0
    JZ  no_init_data
    MOV CX,AX


; RAM Destination address
    MOV AX,SEG _bdata
    MOV ES,AX               ; Destination ES:[DI]
    MOV DI,0                ; Start of initialized variable area in RAM

; ROM Source address
    MOV AX,SEG _etext       ; Source DS:[SI]
    MOV DS,AX               ; Start of initializer storage in ROM
    MOV SI,0

    REP MOVSB               ; Begin byte transfer from ROM to RAM

    mov     al, 'i'
    transmit
```

```
no_init_data:

; Clear uninitialized data area in DGROUP group

    mov     al, 'U'
    transmit

    MOV CX,OFFSET DGROUP:_end    ; End of 'BSS' class in RAM
    MOV DI,OFFSET DGROUP:_edata  ; Start of 'BSS' class in RAM
    SUB CX,DI                    ; Size of 'BSS' class in bytes
    JCXZ no_uninit_data

    MOV AX,0                     ; Initialize to 0
    REP STOSB

    mov     al, 'u'
    transmit


no_uninit_data:
;   Initialize FAR_DATA data in RAM with initializers stored in ROM

;        Transfer Count
    MOV AX,SEG _bfdata
    MOV CX,SEG _efdata
    SUB CX,AX            ; Compute size of FAR_DATA segments in paragraphs
    JCXZ loopend         ; No FAR_DATA class
    MOV DX,CX            ; Saves transfer count in paragraphs
;        Destination
    MOV ES,AX            ; Destination ES:[DI]
    MOV DI,0             ; Start of FAR_DATA class in RAM
;        Source
    MOV AX,SEG _etext    ; Source DS:[SI]
    MOV DS,AX            ; Start of FAR_DATA initializer storage in ROM
    MOV SI,OFFSET DGROUP:_edata      ; _edata is paragraph aligned
;        Normalize Source Pointer
    MOV AX,SI            ; Process base of source pointer
    MOV CL,4
    SHR AX,CL            ; Divide by 16
    MOV BX,AX
    MOV AX,DS
    ADD AX,BX
    MOV DS,AX            ; Adjust base of source pointer
    MOV SI,0             ; Offset of source pointer is zero
    MOV AX,DX            ; Restore transfer count in paragraphs
loopbegin:
    CMP AX,1000H         ; More than 64K bytes to transfer?
    JBE lastxfer         ; No
    MOV CX,8000H         ; Prepare to transfer 8000H words
    SUB AX,1000H         ;
    JMP SHORT xferbegin
lastxfer:
    MOV CL,3
    SHL AX,CL            ; Number of WORDs = paragraph * 8
    MOV CX,AX            ; Set up transfer count in terms of WORDs
    MOV AX,0             ; No more to transfer
xferbegin:
REP     MOVSW                    ; Transfer WORDs from ROM to RAM
    CMP AX,0             ; Any more data to transfer?
    JE loopend           ; No
    ; Adjust Source and Destination pointers
    MOV BX,AX            ; Saves transfer count
    MOV AX,DS
    ADD AX,1000H
    MOV DS,AX
    MOV AX,ES
    ADD AX,1000H
    MOV ES,AX
    MOV SI,0
    MOV DI,0
    MOV AX,BX            ; Restores transfer count
    JMP loopbegin
loopend:


;   Clear uninitialized data area in FAR_BSS class
;        Transfer Count
    MOV AX,SEG _bfbss
    MOV CX,SEG _efbss
    SUB CX,AX            ; Compute size of FAR_BSS segments in paragraphs
    JCXZ loopfend        ; No FAR_BSS class
;        Destination
    MOV ES,AX            ; Destination ES:[DI]
```

234

```
        MOV DI,0              ; Start of FAR_BSS class in RAM
;         Transfer Count
        MOV AX,CX
loopfbegin:
        CMP AX,1000H          ; More than 64K bytes to initialize?
        JBE lastfxfer         ; No
        MOV CX,8000H          ; Prepare to transfer 8000H words
        SUB AX,1000H          ;
        MOV BX,AX             ; Saves transfer count
        JMP SHORT xferfbegin
lastfxfer:
        MOV CL,3
        SHL AX,CL             ; Number of WORDs = paragraph * 8
        MOV CX,AX             ; Set up transfer count in terms of WORDs
        MOV AX,0              ; No more to transfer
        MOV BX,AX             ; Saves transfer count
xferfbegin:
        MOV AX,0
REP     STOSW                 ; Initialize WORDs to zero
        MOV AX,BX             ; Restore transfer count
        CMP AX,0              ; Any more data to transfer?
        JE loopfend           ; No
        ; Adjust Destination pointers
        MOV AX,ES
        ADD AX,1000H
        MOV ES,AX
        MOV DI,0
        MOV AX,BX             ; Restore transfer count
        JMP loopfbegin
loopfend:


;    Clear uninitialized data area in HUGE_BSS class
;         Transfer Count
        MOV AX,SEG _bhbss
        MOV CX,SEG _ehbss
        SUB CX,AX             ; Compute size of HUGE_BSS segments in paragraphs
        JCXZ loophend         ; No HUGE_BSS class
;         Destination
        MOV ES,AX             ; Destination ES:[DI]
        MOV DI,0              ; Start of HUGE_BSS class in RAM
;         Transfer Count
        MOV AX,CX
loophbegin:
        CMP AX,1000H          ; More than 64K bytes to initialize?
        JBE lasthxfer         ; No
        MOV CX,8000H          ; Prepare to transfer 8000H words
        SUB AX,1000H          ;
        MOV BX,AX             ; Saves transfer count
        JMP SHORT xferhbegin
lasthxfer:
        MOV CL,3
        SHL AX,CL             ; Number of WORDs = paragraph * 8
        MOV CX,AX             ; Set up transfer count in terms of WORDs
        MOV AX,0              ; No more to transfer
        MOV BX,AX             ; Saves transfer count
xferhbegin:
        MOV AX,0
REP     STOSW                 ; Initialize WORDs to zero
        MOV AX,BX             ; Restore transfer count
        CMP AX,0              ; Any more data to transfer?
        JE loophend           ; No
        ; Adjust Destination pointers
        MOV AX,ES
        ADD AX,1000H
        MOV ES,AX
        MOV DI,0
        MOV AX,BX             ; Restore transfer count
        JMP loophbegin
loophend:

PUBLIC _init_done
_init_done:

; ****************************************************************************
; Initialize the interrupt vector table here
; ****************************************************************************
; Interrupt types 0 to 4 are dedicated internal interrupts.
; Type 0 - Divide-error
; Type 1 - Single-step
; Type 2 - Non-maskable interrupt
; Type 3 - 1-byte INT instruction or Breakpoint
; Type 4 - Overflow
; Interrupt types 5 to 31 are reserved internal interrupts.
```

```
; Interrupt types 32 to 255 are available for use.
;
; For example:
;   The interrupt handling routine for vector 32 decimal is assumed to
;   be the C function: void interrupt far int_hdlr().
;   The statement declaring code label _int_hdlr as an external far procedure
;   has to be placed outside of all SEGMENT/ENDS pairs.
;   See the sample EXTRN statement above.  It is behind the GROUP statement.
;   Below is the sample code to initialize an entry in the vector table:
;
;        TYPE32 EQU 32
;        MOV AX,0
;        MOV ES,AX          ;Reference base of interrupt vector table
;        MOV AX,OFFSET _int_hdlr
;        MOV ES:WORD PTR TYPE32*4,AX    ;Offset portion of vector 32
;        MOV AX,SEG _int_hdlr
;        MOV ES:WORD PTR TYPE32*4+2,AX ;Base portion of vector 32
;


;    TYPE0 EQU 0
;    MOV AX,0
;    MOV ES,AX   ;Reference base of interrupt vector table
;    MOV AX,OFFSET __cintDIV
;    MOV ES:WORD PTR TYPE0*4,AX    ;Offset portion of vector 0
;    MOV AX,SEG __cintDIV
;    MOV ES:WORD PTR TYPE0*4+2,AX ;Base portion of vector 0


; Setup data and stack segment here before releasing control to _main

      mov      al, 'S'
      transmit

public setup_main
setup_main:
      mov      ax, DGROUP
      mov      ds, ax                               ;Setup data segment
      ASSUME   ds:DGROUP

      mov      ss, ax                               ;Setup stack pointer
      mov      sp, OFFSET DGROUP:stack_top
      ASSUME   ss:DGROUP

;FP emulator init function
      mov      al, 'F'
      transmit
EXTRN         __alinit:FAR
      CALL FAR PTR __alinit
      mov      al, 'p'
      transmit


;Reset the EDAC Hard and Soft Error Interrupts
      mov      dx, PCB_PPI_Ctrl
      mov      al, PCB_PortC_Reset2    ;EDAC Error Acknowledge (Active LOW)
      out      dx, al
      mov      al, PCB_PortC_Set2
      out      dx, al

;Send non-specific EOI to Interrupt Controller
INT_EOI       equ      0FF22h
      mov      dx, INT_EOI
      mov      ax, 08000h
      out      dx, ax

;* Mask ON/OFF Interrupts to the Interrupt Controller
      mov      dx, IMASK
      mov      ax, 0EEh     ;allow INT0, Timers
      out      dx, ax



;Ready to transfer control to the C run-time, enable interrupts.
      mov      al, 'M'
      transmit

      in       al, SCCB_CMD    ;reset pointer
      mov      al, 010h
      out      SCCB_CMD, al    ;WR0: Reset EXT/Status Interrupts
      mov      al, 028h
      out      SCCB_CMD, al    ;WR0: Reset TxINT/Pending
      mov      al, 030h
      out      SCCB_CMD, al    ;WR0: Error Reset
```

```
        mov     al, 038h
        out     SCCB_CMD, al        ;WR0: Reset Highest IUS

PUBLIC  before_main
before_main:

        sti
        call_main                   ;Pass control to C main() in module dcs.c


PUBLIC  _exit, __exit
_exit   LABEL FAR
__exit  LABEL FAR
dcs_halt:
        jmp     dcs_halt


;******************************************************************************
;*
;*  startup_error
;*
;*  This routine halts the processor after displaying the error message which
;*  consists of an exclamation point (!) followed by a number/letter.
;*
;*  Error Codes:
;*      '1'     EDAC RAM Test of 55AA failed
;*     .'2'     EDAC RAM Test of Mod-257 failed
;*
;******************************************************************************

startup_error:

        mov     cl, al              ; error code
        mov     al, '!'
        transmit
        mov     al, cl
        transmit
        mov     al, '@'
        transmit

;First ES
        mov     ax, es
        mov     cl, 12
        display

        mov     ax, es
        mov     cl, 8
        display

        mov     ax, es
        mov     cl, 4
        display

        mov     ax, es
        display

        mov     al, ':'
        transmit

;Now DI
        mov     ax, di
        mov     cl, 12
        display

        mov     ax, di
        mov     cl, 8
        display

        mov     ax, di
        mov     cl, 4
        display

        mov     ax, di
        display

sehlt:
        jmp     sehlt



;*
;* FP Support
;*
```

237

```
; Trap for missing floating-point software.
; The floating point initialization routine (__fpmath) will call this routine
; when one of the following conditions occurs:
; (1) 8087 floating point library (87.lib) is linked in but no 8087 coprocessor
;     is present, that is, floating point emulator library is not linked.
; (2) Floating point i/o conversions are done, but no floating-point variables
;     or expressions are used in the program.
; Default action is to halt the processor.
      PUBLIC __fptrap
__fptrap LABEL FAR
      MOV AX,3        ; Identify label __fptrap.
;        HLT
      JMP __fptrap


; ERROR HANDLING
;
; INT 21H called. Register AH contains the function code.
; If function code is 00H, 25H, 35H or 4CH, the interrupt handler
; will process the call.
; For all other function codes, the interrupt handler will jump here.
; Default action is to halt the processor.
      PUBLIC __doscalled
__doscalled LABEL FAR
;
; If you want to ignore the dos call,
; replace the following instructions.
;        MOV AX,4        ; Identify label __doscalled
;        HLT
;        JMP __doscalled
; WITH:
      EXTRN __ignored:FAR
      JMP FAR PTR __ignored
; This will cause the interrupt handler to ignore the dos call.
; __ignore is an entry point back to the interrupt handler.
;


; PROCEDURE __cintDIV:
;
      PUBLIC __cintDIV
__cintDIV PROC FAR      ; Divide by 0 interrupt handler.
      MOV AX,5        ; User-defined error recovery routine.
;        HLT             ; Identify label __cintDIV.
      JMP __cintDIV
__cintDIV ENDP


; VARIABLE _errno:
;
; For certain C run-time functions, when an error condition occurs within the
; function, an error code will be placed in the _errno global variable.
; If a function sets the _errno variable upon error, its reference page will
; explicitly mention the _errno variable.
; All of the error codes are described in the Microsoft C run-time library
; reference manual.
; The values of these error codes are listed in the errno.h include file.


; FUNCTION matherr:
;
; You may supply your own version of matherr function in your C program.
; If you do, you can obtain a value from the type field of the exception
; data structure which corresponds to the math error code listed in the
; math.h include file.  the Microsoft C run-time library reference
; manual contains a detail description of the matherr fucntion and the
; math error codes.
;
; If you do not link in your own matherr function, the matherr function
; included in the Microsoft C run-time library will be linked in.
; The function simply returns a zero value.
;
; If you link in your own version of the matherr function, it may perform
; special error handling.  if corrective action is taken and the
; the return value should be nonzero.


; PROCEDURE __FF_MSGBANNER:
;
; The __FF_MSGBANNER procedure will be called when an error condition occurs
; within in a math function and certain C run-time functions.
; It writes the first part of run-time error messages to standard
; error as follows:
; '\r\nrun-time error '.
```

```
; It is implemented as a null procedure here.
;
      PUBLIC __FF_MSGBANNER
__FF_MSGBANNER PROC NEAR
      RET
__FF_MSGBANNER ENDP


; PROCEDURE __wrt2err:
;
; The __wrt2err procedure will be called when an error condition occurs
; within in a math function and certain C run-time functions.
; It takes a near pointer in BX (DS:BX) which points to a LSTRING which is
; to be written to standard error.  A LSTRING is a one-byte length followed
; by that many bytes for the character string (as opposed to a null-
; terminated string).
; These LSTRINGs has the form of the first character being a capital letter
; followed by four digits.  For examples, 'R6001', 'M6101', etc.  The
; meaning of these error numbers are explained in detail in the Microsoft C
; reference manual.
; The __wrt2err procedure is implemented as a null procedure here.
;
      PUBLIC __wrt2err
__wrt2err PROC NEAR
;         DS:BX points to the LSTRING.
      RET
__wrt2err ENDP


; PROCEDURE __NMSG_WRITE:
;
; The  __NMSG_WRITE procedure will be called when an error condition occurs
; within in a math function and certain C run-time functions.
; It searches the MSG segment for the address of a message string corresponding
; to the error condition.
; If a message string is found, DS:DX = string address, CX = string length.
; You may process or ignore the error message string.
;
; The follow table lists some of the error message numbers and the message strings:
; 253   ': MATH',13,10,'- floating-point error: ',0
; 101   'invalid',13,10,0
; 102   'denormal',13,10,0
; 103   'divide by 0',13,10,0
; 104   'overflow',13,10,0
; 105   'underflow',13,10,0
; 106   'inexact',13,10,0
; 107   'unemulated',13,10,0
; 108   'square root',13,10,0
; 109   13,10,0
; 110   'stack overflow',13,10,0
; 111   'stack underflow',13,10,0
; 112   'explicitly generated',13,10,0
;
      PUBLIC __NMSG_WRITE
__NMSG_WRITE PROC NEAR
      PUSH BP
      MOV BP,SP
      PUSH    DS
      POP     ES
      MOV DX,WORD PTR [BP+4]   ; DX = error message number
      CMP     DX,253
      JE      NOTFOUND         ; Skip error message no. 253,
                        ; But process other error message numbers
      ASSUME DS:DGROUP
      MOV     SI,OFFSET DGROUP:MSG   ; start of near messages
TLOOP:
      LODSW                    ; AX = current message number
      CMP     AX,DX
      JE      FOUND            ; Found error message string
      INC     AX
      XCHG    AX,SI
      JZ      NOTFOUND         ; At end and error message string not found
      XCHG    DI,AX
      XOR     AX,AX
      MOV     CX,-1
      REPNE   SCASB            ; Skip until 0H
      MOV     SI,DI
      JMP     TLOOP            ; Try next entry
FOUND:
      XCHG    AX,SI            ; SI = offset to string address
      XCHG    DX,AX            ; DS:DX = string address
      MOV     DI,DX            ; Determine length of message string
      XOR     AX,AX            ; String is terminated with byte 0h
      MOV     CX,-1
```

239

```
        REPNE   SCASB           ; ES = DS already
        NOT     CX
        DEC     CX              ; CX = string length
; May include user-defined code here to output error message
; DS:DX = string address, CX = string length
NOTFOUND:
        MOV SP,BP
        POP BP
        RET
__NMSG_WRITE ENDP
;
        PUBLIC __dataseg
__dataseg DW DGROUP



_TEXT   ENDS



EMULATOR_TEXT  segment para public 'CODE'
        public __EmDataSeg
__EmDataSeg dw  EMULATOR_DATA
EMULATOR_TEXT  ends



BOOTSTRAP SEGMENT AT 0FFFFH
        cli

        mov             dx, UMCS
        mov             ax, UMCS_DATA    ;Upper Memory CS start of EPROM F000:0, 64K
        out             dx, ax

;       mov     dx, 0FFA0h
;       mov     ax, 0C038h
;       out     dx, ax

        jmp             far ptr START_

        db              'Jah'

BOOTSTRAP ENDS


;*******************************************************************************
;*
;*   C_ETEXT
;*
;*   If you specify the INITDATA control in the PROM86, v5.2 and up, it will
;*   cause PROM86 to place all initializers in ROM starting at this
;*   location.  The start-up routine assumes these initializers are placed
;*   behind program code in ROM and copy them to RAM at power-up.
;*******************************************************************************

C_ETEXT SEGMENT PARA PUBLIC 'CODE_END'

PUBLIC _etext
_etext  LABEL BYTE      ; This label marks the end of program code.

C_ETEXT ENDS



END START_      ; Make sure the START_ symbol is here!
```

End of startup.asm

stpi.h, stpi.c

```
/****************************************************************************
 *
 *   STPI.H
 *
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who     What
 *   ---------------+-------+----------------------------------------------
 *   3 Nov 1993      Jah     Creation
 *
 ****************************************************************************/

#ifdef STPI
        #define         CMD_STR_LEN       80

        typedef struct
        {
            char name[10];
            void (*fptr)(char *inbuf);
            char usage[81];

        } cmd_struct;

        static char     *get_token(char *buf, char *token);
        static void     parse_cmd(char *cptr);
        static char     *skip_blanks(char *cptr);

        static void     clear_screen(char *cptr);
        static void     in_port(char *cptr);
        static void     in_portw(char *cptr);
        static void     out_port(char *cptr);
        static void     out_portw(char *cptr);
        static void     pcbr(char *cptr);
        static void     pcbw(char *cptr);

        static void     ad_config(char *cptr);
        static void     ad_int(char *cptr);
        static void     ad_read(char *cptr);
        static void     ad_status(char *cptr);

        static void     debug_cmd(char *cptr);

        static void     disp_b(unsigned char a);
        static void     disp_w(unsigned int a);

        static void     m_on(char *cptr);
        static void     m_off(char *cptr);
        static void     m_clear(char *cptr);
        static void     m_spread(char *cptr);
        static void     m_hunt(char *cptr);
        static void     m_scca(char *cptr);
        static void     m_sccb(char *cptr);

        static void     pa_read(char *cptr);
        static void     pa_write(char *cptr);

        static void     edit(char *cptr);
        static void     dump(char *cptr);
        static void     load(char *cptr);
        static void     goto_load(char *cptr);

        static void     msa_cmd(char *cptr);
        static void     msb_cmd(char *cptr);

        static void     time_cmd(char *cptr);

        static void     test_cmd(char *cptr);

        static void     rf_cmd(char *cptr);

        static void     tx_cmd(char *cptr);
        static void     rx_cmd(char *cptr);
```

```c
        static void        eps_cmd(char *cptr);

        static void        tmux_cmd(char *cptr);

        static void        tlm_cmd(char *cptr);

        static void        shutdown_cmd(char *cptr);

        static void        bcm_cmd(char *cptr);
#endif


#ifndef STPI
        extern void monitor(void);
#endif




/*****************************************************************************
 *
 *   STPI.C
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   Date            Who    What
 *   ---------------+-------+---------------------------------------------
 *   3 Nov 1993      Jah    Creation
 *
 *****************************************************************************/

#include <string.h>
#include <ctype.h>
#include <math.h>


#include "gen_defs.h"

#define        STPI
#include "stpi.h"
#undef         STPI

#include "ad.h"
#include "bcm.h"
#include "clock.h"
#include "dcs.h"
#include "eps.h"
#include "pcb.h"
#include "print.h"
#include "modem.h"
#include "tlm.h"         /* must be before msu.h */
#include "msu.h"
#include "scc.h"
#include "terms.h"


WORD adr(int ch);


static cmd_struct cmd_table[] =
{
    "cls",         clear_screen,      "CLS                              Clear the screen.",
    "help",        parse_cmd,         "HELP                             Display this menu.",

    "adc",         ad_config,         "ADC                              Show A/D configuration.",
    "adi",         ad_int,            "ADI                              Show A/D interrupt information.",
    "adr",         ad_read,       "ADR    <channel>             Read A/D channels.",
    "ads",         ad_status,         "ADS                              Show A/D status.",

    "debug", debug_cmd,         "DEBUG <0/1>                      Debug info Off/Onn",

    "in",          in_port,       "IN     <port>                Input byte from port.",
    "inw",         in_portw,          "INW    <port>                Input word from port.",
    "out",         out_port,          "OUT    <port>    <value>        Output byte from port.",
    "outw",        out_portw,         "OUTW   <port>    <value>        Output word from port.",

    "pcbr",        pcbr,              "PCBR   <select> <addr>          PCB read.",
    "pcbw", pcbw,                 "PCBW   <select> <addr> <value>  PCB write.",
```
242

```
        "m0",           m_off,          "M0                                      Modem OFF.",
        "m1",           m_on,           "M1                                      Modem ON.",
        "c",      m_clear,      "C                                  Modem clear mode 78.125k.",
        "hunt",         m_hunt,         "HUNT                                    Enter Hunt Mode.",
        "s",      m_spread,       "S                                  Modem spread mode 9.842k.",
        "scca",         m_scca,         "SCCA  <WR#> <data>              Send data to write register (A).",
        "sccb",         m_sccb,         "SCCB  <WR#> <data>              Send data to write register (B).",

        "par",          pa_read,      "PAR                              PA-100 read registers.",
        "paw",          pa_write,       "PAW   <filename>                 PA-100 write registers.",

        "load",         load,           "LOAD  <filename>                 Load binary image to 1000:0100.",
        "goto",         goto_load,      "GOTO                             Jump to 1000:0100.",
        "dump",         dump,           "DUMP  <seg> <off>                Dump memory.",
        "edit",         edit,           "EDIT  <seg> <off>                Edit memory locations.",

        "msa",          msa_cmd,      "MSA   <0/1;r/w/e|s/f> <addr> <data> Mass Storage A Control.",
        "msb",          msb_cmd,      "MSB   <0/1:r/w/e:s/f> <addr> <data> Mass Storage B Control.",

        "time",         time_cmd,       "TIME  <YY:MM:DD:HH:MM:SS>             Get/Set time.",

        "test",         test_cmd,       "TEST",

        "rf",           rf_cmd,         "RF <0/1; T/R; Tx/Rx; LOP/LOA; LHP/LHA; P #; LNA0/LNA1; HPA0/HPA1; E #",

        "tx",           tx_cmd,         "TX <text message>",                                      •
        "rx",           rx_cmd,         "RX <I; S>",

        "eps",          eps_cmd,      "EPS <B A/B C/D/O/T ON/OFF; C sys ON/OFF; V A#/B#/S; I A/B/S/P#; W>",

        "tmux",         tmux_cmd,       "TMUX <A/B CH#/ON/OFF>",

        "bcm",          bcm_cmd,      "BCM <ON/OFF>",

        "shutdown", shutdown_cmd, "SHUTDOWN",

        "tlm",          tlm_cmd,        "    ",

        "",             parse_cmd,        ""
};


/******************************************************************************
 *
 *   monitor()
 *
 ******************************************************************************/

void monitor(void)
{
        static char     cstr[CMD_STR_LEN + 1];
        char            *cptr = cstr;
        char            c;
        static int      i = 0;
        static int      end_string = FALSE;



        while (is_serial_in() && !end_string)
        {
            c = get_char();
            if ((i < CMD_STR_LEN) && (c != CR))
                cstr[i++] = c;

            if (c == CR)
            {
                cstr[i] = NULL_CHAR;       /* replaces CR with a NULL_CHAR */
                end_string = TRUE;
            }

        }  /* End of WHILE */


        if (end_string)
        {
            parse_cmd(cptr);
            serial_out(CTRL_W);        /* indicate end of processing command */

            i = 0;                     /* reset pointer into command string to zero */
            end_string = FALSE;        /* allow new string building next time */
        }

}  /* End of monitor() */
```
243

```
/***************************************************************************
 *
 *   parse_cmd()
 *
 ***************************************************************************/

void parse_cmd(char *cptr)
{
    int      n, m;
    char cmd[10];
    int      found;
    static char  last_cmd[80];


    if (cptr[0] == '!')
        strcpy(cptr, last_cmd);        /* copy last command to this command */
    else
        strcpy(last_cmd, cptr);        /* otherwise, save this command for next time */

    cptr = get_token(cptr, cmd);

    for (found = FALSE, n = 0; cmd_table[n].name[0] != '\0'; n++)
    {
        if (stricmp(cmd, cmd_table[n].name) == 0)
        {
            found = TRUE;
            if (stricmp(cmd, "help") == 0)
            {
                home();
                clr();
                dprint("PANSAT Monitor Commands\n");
                dprint("=======================\n");
                for (m = 0; cmd_table[m].name[0] != '\0'; m++)
                    dprint("%s\n", cmd_table[m].usage);

            }

            else if (cmd_table[n].name[0] != '\0')
            {
                dprint("\n");
                (*cmd_table[n].fptr)(cptr);
                dprint("\n");
            }

            break;
        }

    }

    if (!found)
    {
        serial_out(0x07);/* beep for error */
        dprint("Command error.\n");
    }

}  /* End of parse_cmd()*/


/***************************************************************************
 *
 *   get_token()
 *
 ***************************************************************************/

char *get_token(char *buf, char *token)
{
    if (*buf == NULL_CHAR)
    {
        *token = NULL_CHAR;
        return(buf);
    }

    while ((*buf != ' ') && (*buf != NULL_CHAR))
        *token++ = *buf++;
    *token = NULL_CHAR;

    if (*buf == NULL_CHAR)
        return(buf);
    else
        skip_blanks(buf);
```

```
}   /* End of get_token() */


/****************************************************************************
 *
 *   skip_blanks()
 *
 ***************************************************************************/

char *skip_blanks(char *buf)
{
    while ((*buf == ' ') && (*buf != NULL_CHAR))
        buf++;

    return(buf);

}   /* skip_blanks() */


/****************************************************************************
 ****************************************************************************
 *
 *        Supported monitor commands
 *
 ****************************************************************************
 ****************************************************************************/


/****************************************************************************
 *
 *   clear_screen()
 *
 ***************************************************************************/

void clear_screen(char *cptr)
{
    home();
    clr();

}   /* End of clear_screen() */


/****************************************************************************
 *
 *   in_port()
 *
 ***************************************************************************/

void    in_port(char *cptr)
{
    char            param[20];
    unsigned char    value;


    cptr = get_token(cptr, param);
    value = inp(cnv_hex(param));

    dprint("Port %X = %X   ", cnv_hex(param), value);
    disp_b(value);

}   /* End of in_port() */


/****************************************************************************
 *
 *   in_portw()
 *
 ***************************************************************************/

void    in_portw(char *cptr)
{
    char            param[20];
    unsigned int value;

    cptr = get_token(cptr, param);
    value = inpw(cnv_hex(param));

    dprint("Port %X = %X   ", cnv_hex(param), value);
    disp_w(value);

}   /* End of in_portw() */


/****************************************************************************
```

```
 *
 *   out_port()
 *
 ****************************************************************************/

void    out_port(char *cptr)
{
    char port[10], value[10];


    cptr = get_token(cptr, port);
    cptr = get_token(cptr, value);

    dprint("Out %X to port %X", cnv_hex(value), cnv_hex(port));

    outp(cnv_hex(port), (BYTE)cnv_hex(value));

}   /* End of out_port() */


/****************************************************************************
 *
 *   out_portw()
 *
 ****************************************************************************/

void    out_portw(char *cptr)
{
    char port[10], value[10];


    cptr = get_token(cptr, port);
    cptr = get_token(cptr, value);

    dprint("Out %X to port %X", cnv_hex(value), cnv_hex(port));

    outpw(cnv_hex(port), (WORD)cnv_hex(value));

}   /* End of out_portw() */


/****************************************************************************
 *
 *   pcbr
 *
 ****************************************************************************/

void pcbr(char *cptr)
{
    char            cbuf[20];
    unsigned int select, addr, value;


    cptr = get_token(cptr, cbuf);
    select = cnv_hex(cbuf);

    cptr = get_token(cptr, cbuf);
    addr = cnv_hex(cbuf);

    value = pcb_read(select, addr);

    dprint("PCBR %x %x %x", select, addr, value);

}   /* End of pcbr() */


/****************************************************************************
 *
 *   pcbw
 *
 ****************************************************************************/

void pcbw(char *cptr)
{
    char            cbuf[20];
    unsigned int select, addr, value;


    cptr = get_token(cptr, cbuf);
    select = cnv_hex(cbuf);

    cptr = get_token(cptr, cbuf);
    addr = cnv_hex(cbuf);
```

246

```c
        cptr = get_token(cptr, cbuf);
        value = cnv_hex(cbuf);

        pcb_write(select, addr, value);

        dprint("PCBW %x %x %x", select, addr, value);

}   /* End of pcbw() */


/******************************************************************************
 *
 *   disp_b()
 *
 ******************************************************************************/

void disp_b(unsigned char a)
{
        int  n;


        for (n = 0; n <= 7; n++)
        {
            if (n == 4)
                dprint(" ");

            if (a & 0x80)
                dprint("1");
            else
                dprint("0");

            a = a << 1;
        }

}   /* End of disp_b() */


/******************************************************************************
 *
 *   disp_w()
 *
 ******************************************************************************/

void disp_w(unsigned int a)
{
        int  n;


        for (n = 0; n <= 15; n++)
        {
            if (n%4 == 0)
                dprint(" ");

            if (a & 0x8000)
                dprint("1");
            else
                dprint("0");

            a = a << 1;
        }

}   /* End of disp_w() */



#define  AD_BASE       0x80
#define  AD_INSTR0     AD_BASE
#define  AD_INSTR1     AD_BASE + 2
#define  AD_INSTR2     AD_BASE + 4
#define  AD_INSTR3     AD_BASE + 6
#define  AD_INSTR4     AD_BASE + 8
#define  AD_INSTR5     AD_BASE + 0x0A
#define  AD_INSTR6     AD_BASE + 0x0C
#define  AD_INSTR7     AD_BASE + 0x0E
#define  AD_CONFIG     AD_BASE + 0x10
#define  AD_IER        AD_BASE + 0X12
#define  AD_ISR        AD_BASE + 0x14
#define  AD_TIMER AD_BASE + 0x16
#define  AD_FIFO       AD_BASE + 0x18
#define  AD_LIMIT AD_BASE + 0x1A

/* Masks */
#define  RAM00         0x0000
#define  RAM01         0x0100
```

```c
#define RAM02        0x0200

/*****************************************************************************
 *
 *   ad_config()
 *
 *****************************************************************************/

void ad_config(char *cptr)
{
    unsigned int temp = inpw(AD_CONFIG);


    dprint("A to D Configuration Register: %X = %X\n", AD_CONFIG, temp);

    dprint("    Start = ");
    if (temp & 0x0001)
        dprint("1, Sequencer is running.\n");
    else
        dprint("0, Sequencer is stopped.\n");

    dprint("    Reset = ");
    if (temp & 0x0002)
        dprint("1, unit is still resetting.\n");
    else
        dprint("0, unit is not resetting.\n");

    dprint("    Auto Zero = ");
    if (temp & 0x0004)
        dprint("1, in progress.\n");
    else
        dprint("0, not occurring.\n");

    dprint("    Full Calibration = ");
    if (temp & 0x0008)
        dprint("1, in progress.\n");
    else
        dprint("0, not occurring.\n");

    dprint("    Standby = ");
    if (temp & 0x0010)
        dprint("1, in standby mode.\n");
    else
        dprint("0, not in standby mode.\n");

    dprint("    Channel Mask = ");
    if (temp & 0x0020)
        dprint("1, FIFO bits 15-13 are sign.\n");
    else
        dprint("0, FIFO bits 15-13 are pointer.\n");

    dprint("    Short Auto Zero = ");
    if (temp & 0x0040)
        dprint("1, occurs before every conversion.\n");
    else
        dprint("0, disabled.\n");

    dprint("    Sync = ");
    if (temp & 0x0080)
        dprint("1, SYNC pin is an output.\n");
    else
        dprint("0, SYNC pin is an input.\n");

    dprint("    RAM pointer = %X\n", (temp & 0x0300) >> 8);

    dprint("    Test = ");
    if (temp & 0x0400)
        dprint("1, in test mode.\n");
    else
        dprint("0, not in test mode.\n");

    dprint("    Diagnostic = ");
    if (temp & 0x0800)
        dprint("1, in diagnostic mode.\n");
    else
        dprint("0, not in diagnostic mode.\n");

}   /* End of adc() */


/*****************************************************************************
 *
 *   ad_int()
 *
```

```c
 ****************************************************************************/

void ad_int(char *cptr)
{
    unsigned int ier = inpw(AD_IER);
    unsigned int temp, i;


    dprint("A to D Interrupts: %X = %X\n", AD_IER, ier);

    dprint("    Interrupts enabled: ");
    for (temp = ier, i = 0; i <= 7; i++)
    {
        if (i != 6)
            if (temp & 0x0001)
                dprint("%d  ", i);
        temp >> 1;
    }
    dprint("\n");
    dprint("    Sequencer address to generate INT1 = %X\n", (ier & 0x0F00) >> 8);
    dprint("    # of conversions in FIFO to generate INT2 = %X\n", (ier & 0xF800) >> 11);

}   /* End of ad_int() */


/*****************************************************************************
 *
 *   ad_read
 *
 ****************************************************************************/

void ad_read(char *cptr)
{
    unsigned int c, value, isr, temp;
    double       x;
    char         buf[20];


    outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)   /* Wait for Reset bit to clear */
        ;

    outpw(AD_CONFIG, 0x0008);          /* Full Calibration */
    while (inpw(AD_CONFIG) & 0x0008)   /* Wait for calibration to finish */
        ;

    outpw(AD_CONFIG, 0x0000);          /* Stop sequencer, point to RAM 00 */


    /* DCS Temperature, MUX+ = IN0, MUX- = GND */
    outpw(AD_INSTR0, 0xF202);     /* acq. time = full way, no wdog, 12-bit, */
                                  /*   Timer ON, NO sync, Vin- = Gnd, Vin+ = IN0 */
                                  /*   Pause = YES, loop = NO */
                                  /*   Note: pause will happen after loop */

    /* Modem Temperature, MUX+ = IN1, MUX- = GND */
    outpw(AD_INSTR1, 0xF204);     /* acq. time = full way, no wdog, 12-bit, */
                                  /*   Timer ON, NO sync, Vin- = Gnd, Vin+ = IN1 */
                                  /*   no pause, loop = NO */

    /* TMUXA, MUX+ = IN4, MUX- = GND */
    outpw(AD_INSTR2, 0xF210);     /* acq. time = full way, no wdog, 12-bit, */
                                  /*   Timer ON, NO sync, Vin- = Gnd, Vin+ = IN4 */
                                  /*   no pause, loop = NO */

    /* TMUXB, MUX+ = IN6, MUX- = GND */
    outpw(AD_INSTR3, 0xF218);     /* acq. time = full way, no wdog, 12-bit, */
                                  /*   Timer ON, NO sync, Vin- = Gnd, Vin+ = IN6 */
                                  /*   no pause, loop = NO */

    /* EPS, MUX+ = IN2, MUX- = GND */
    outpw(AD_INSTR4, 0xF209);     /* acq. time = full way, no wdog, 12-bit, */
                                  /*   Timer ON, NO sync, Vin- = Gnd, Vin+ = IN2 */
                                  /*   no pause, loop = YES */

    /* RAM 01(1) and 10(2) are not set - limit stuff */


    /* Timer to slow the conversion rate */
/*  outpw(AD_TIMER, 0x1000); */
    outpw(AD_TIMER, 2000);

    outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)   /* Wait for Reset bit to clear */
```

249

```c
                ;
    outpw(AD_CONFIG, 0x0001); /* start sequencer */

    /* Wait for INT 5 - Pause Interrupt */
    while (!inpw(AD_ISR & 0x0020))
        ;
    /* Sequencer is stopped, due to Pause in last instruction */

    /* Wait for 5 samples in the FIFO */
    while (((inpw(AD_ISR) & 0xF800) >> 11) < 5)
        ;

    c = (inpw(AD_ISR) & 0xF800) >> 11;
    while (c)
    {
        value = inpw(AD_FIFO);
        dprint("%u) ", (value & 0xE000) >> 13);
        if (value & 0x1000)
            dprint("-");
        dprint("%u, ", (value & 0x0FFF));

        dprint("0x%X, ", (value & 0x0FFF));

        x = (double)(value & 0x0FFF);
        x = (x/4095.0)*5.0;

        dprint("%3.3lf Volts", x);

        switch((value&0xE000)>>13)
        {
            case 0:  /* DCS Temp */
                x = (x - 0.5)*100;
                dprint(", DCS Temp. = %3.3lf C", x);
                break;

            case 1:  /* Modem Temp */
                x = (x - 0.5)/.01;
                dprint(", Modem Temp. = %3.3lf C", x);
                break;

            case 2:  /* TMUXA Temp */
                dprint(", TMUXA Temp. = %d C", cnv_therm(value&0x0FFF));
                break;

            case 3:  /* TMUXB Temp */
                dprint(", TMUXB Temp. = %d C", cnv_therm(value&0x0FFF));
                break;

            case 4:  /* EPS Measurement */
                dprint(", EPS");
                break;

        }

        dprint("\n");
        c--;
    }


    /* Do a diagnostic test */

    /* Using Diagnostic mode: VIN+ = 000 = VREFOUT, VIN- = 000 = GND */
    outpw(AD_INSTR0, 0xF202);        /* acq. time = full way, no wdog, 12-bit, */
                                     /*    Timer ON, NO sync */
                                     /*    Pause = YES, loop = NO */
                                     /*    Note: pause will happen after loop */

    /* Using Diagnostic mode: VIN+ = 001 = VREF+, VIN- = 001 = VREF- */
    outpw(AD_INSTR1, 0xF225);        /* acq. time = full way, no wdog, 12-bit, */

    outpw(AD_CONFIG, 0x0002);            /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)     /* Wait for Reset bit to clear */
        ;

    outpw(AD_CONFIG, 0x0801); /* start sequencer in diagnostic mode */

    /* Wait for INT 5 - Pause Interrupt */
    while (!inpw(AD_ISR & 0x0020))
        ;
    /* Sequencer is stopped, due to Pause in last instruction */

    /* Wait for 2 samples in the FIFO */
    while (((inpw(AD_ISR) & 0xF800) >> 11) < 2)
```

250

```c
        ;
    c = (inpw(AD_ISR) & 0xF800) >> 11;
    while (c)
    {
        value = inpw(AD_FIFO);
        if (c == 2)
            dprint("Diagnostic: VREFOUT to GND: ");
        else if (c == 1)
            dprint("Diagnostic: VREF+ to VREF-: ");
        dprint("%u ", (value & 0xE000) >> 13);
        if (value & 0x1000)
            dprint("-");
        dprint("%u, ", (value & 0x0FFF));

        dprint("0x%X, ", (value & 0x0FFF));

        x = (double)(value & 0x0FFF);
        x = (x/4095.0)*5.0;

        dprint("%3.3lf Volts\n", x);
        c--;
    }

}   /* End of ad_read() */


/******************************************************************************
 *
 *   ad_status
 *
 ******************************************************************************/

void ad_status(char *cptr)
{
    unsigned int isr = inpw(AD_ISR);
    unsigned int temp, i;


    dprint("A to D Interrupts: %X = %X\n", AD_ISR, isr);

    dprint("    Interrupts Generated: ");
    for (temp = isr, i = 0; i <= 7; i++)
    {
        if (i != 6)
            if (temp & 0x0001)
                dprint("%d   ", i);
        temp >> 1;
    }
    dprint("\n");
    dprint("    Sequencer's current address =  %X\n", (isr & 0x0F00) >> 8);
    dprint("    # of conversions in FIFO = %X\n", (isr & 0xF800) >> 11);

}   /* End of ad_status() */


/******************************************************************************
 *
 *   m_off()
 *
 ******************************************************************************/

void m_off(char *cptr)
{
    modem_off();

    dprint("Modem OFF");

}   /* End of m_off() */


/******************************************************************************
 *
 *   m_on()
 *
 ******************************************************************************/

void m_on(char *cptr)
{
    modem_on();

    dprint("Modem ON");

}   /* End of m_on() */
```

```
/************************************************************************
 *
 *   m_clear()
 *
 ************************************************************************/

void m_clear(char *cptr)
{
    modem_clear();

    dprint("Modem Clear");

}   /* End of m_clear() */


/************************************************************************
 *
 *   m_spread()
 *
 ************************************************************************/

void m_spread(char *cptr)
{
    modem_spread();

    dprint("Modem Spread");

}   /* End of m_spread() */


/************************************************************************
 *
 *   m_hunt()
 *
 ************************************************************************/

void m_hunt(char *cptr)
{
    scc_hunt();

}   /* End of m_hunt() */


/************************************************************************
 *
 *   m_scca()
 *
 ************************************************************************/

void m_scca(char *cptr)
{
    char reg[10], value[10];
    WORD data;


    cptr = get_token(cptr, reg);
    cptr = get_token(cptr, value);

    if (value[0] == NULL_CHAR)
    {
        outp(SCCA_CMD, cnv_hex(reg));
        data = inp(SCCA_CMD);
        dprint("SCCA RR#%s = %X    ", reg, data);
        disp_b((BYTE)data);
    }

    else
        scca_wreg(cnv_hex(reg), cnv_hex(value));

}   /* End of m_scca() */


/************************************************************************
 *
 *   m_sccb()
 *
 ************************************************************************/

void m_sccb(char *cptr)
{
    char reg[10], value[10];
    WORD data;
```

```
        cptr = get_token(cptr, reg);
        cptr = get_token(cptr, value);

        if (value[0] == NULL_CHAR)
        {
            outp(SCCB_CMD, cnv_hex(reg));
            data = inp(SCCB_CMD);
            dprint("SCCB RR#%s = %X    ", reg, data);
            disp_b((BYTE)data);
        }

        else
            sccb_wreg(cnv_hex(reg), cnv_hex(value));

}   /* End of m_sccb() */


/****************************************************************************
 *
 *   pa_read()
 *
 ****************************************************************************/

void pa_read(char *cptr)
{
    pa100_read_regs();

}   /* End of pa_read() */


/****************************************************************************
 *
 *   pa_write()
 *
 *   This command requests of the terminal emulator program on the other side of
 *   the RS-232 to send the contents of the filename requested via this command
 *   parameter.  The filename is preceeded with a CTRL-Z which indicates to the
 *   terminal emulator that a request to open, read, and download a local file
 *   across the serial port.  The download is complete when a CTRL-Z from the
 *   terminal emulator is sent and received at this end.
 *
 *   The data across the serial line comes in pairs.  The first is the address
 *   offset (into the PA-100), followed by the data to be written to that
 *   address.  If there is no data associated with the filename, then only
 *   the terminating CTRL-Z will be sent, causing the count, c, to be zero.
 *
 ****************************************************************************/

void pa_write(char *cptr)
{
    char    filename[30];
    BYTE    x;
    int     c, i;
    static  pa100_instr_struct pa100_config[75];


    cptr = get_token(cptr, filename);
    dprint("Receiving data from %s.\n", filename);  -

    serial_out((BYTE)CTRL_Z);
    dprint("%s", filename);
    serial_out((BYTE)CTRL_Z);


    if ((c = get_char()) == 0)
        dprint("No data downloaded!");

    else if (c < 75)
    {
        for (i = 0; i < c; i++)
        {
            pa100_config[i].address = get_char();
            pa100_config[i].data = get_char();
        }

        pa100_config[c].address = 0xFF;
        pa100_config[c].data = 0xFF;

        dprint("Data received for %d instructions.\n", c);
        pa100_write_table(pa100_config);
        dprint("Data downloaded to PA100.");
    }
```

253

```c
        else
            dprint("Error, attempting to load a table greater than 75 elements!");

}   /* End of pa_write() */


/******************************************************************************
 *
 *   load()
 *
 ******************************************************************************/

void load(char *cptr)
{
    char            filename[30];
    BYTE            x;
    unsigned int c, i;
    unsigned long   temp;
    BYTE far *      ptr;


    temp = ((unsigned long)0x1000<<16) + (unsigned long)(0x0100);
    ptr = (BYTE far *)temp;


    cptr = get_token(cptr, filename);
    dprint("Loading program image from %s to %P.\n", filename, ptr);

    serial_out((BYTE)CTRL_Y);
    dprint("%s", filename);
    serial_out((BYTE)CTRL_Y);


    c = get_char();
    c *= 256;
    c += get_char();

    if (c == 0)
        dprint("No program image loaded!");

    else
    {
        i = c;
        while (i--)
        {
            x = get_char();
            *ptr = x;
            ++ptr;
        }

        dprint("Program image loaded, %d bytes.", c);

    }

}   /* End of load() */


/******************************************************************************
 *
 *   goto_load()
 *
 ******************************************************************************/

void goto_load(char *cptr)
{
    /* Transfer control to the os RAM image at 0x1000:0100 */
    _asm
    {
        cli
        mov     ax, 0x1000
        push ax                 ; new CS = 0x1000
        mov     ax, 0x0100
        push ax                 ; new IP = 0x0100
        sti
        retf
    }

}   /* End of goto_load() */


/******************************************************************************
 *
 *   dump()
```

254

```
    *
    ****************************************************************************/

void dump(char *cptr)
{
        char            buf[10];
        static BYTE far *      ptr = 0L;
        BYTE            val;
        WORD            segment, offset;
        unsigned long      temp;
        int             i, j;


        cptr = get_token(cptr, buf);
        if (buf[0] != NULL_CHAR)
        {
            segment = cnv_hex(buf);
            cptr = get_token(cptr, buf);
            offset = cnv_hex(buf);
            temp = ((unsigned long)segment<<16) + (unsigned long)(offset & 0xFFFF);
            ptr = (BYTE far *)temp;
        }



        for (i = 0; i < 256; i += 16)
        {
            dprint("%P  ", (BYTE far *)(ptr + i));
            for (j = 0; j < 16; j++)
            {
                if (j == 8)
                    dprint(" ");
                val = ((BYTE)*(ptr+i+j));
                if (val < 0x10)
                    dprint("0");
                dprint("%X ", val);
            }
            dprint("  ");
            for (j = 0; j < 16; j++)
            {
                val = ((BYTE)*(ptr+i+j));
                if ((val >= 32) && (val <= 127))
                    dprint("%c",*(ptr+i+j));
                else
                    dprint(".");
            }
            dprint("\n");
        }

        ptr += 256;

}  /* End of dump() */


/****************************************************************************
 *
 *   edit()
 *
 ****************************************************************************/

void edit(char *cptr)
{
        char            buf[10];
        BYTE far *      ptr;
        BYTE            val;
        WORD            segment, offset;
        unsigned long      temp;


        cptr = get_token(cptr, buf);
        segment = cnv_hex(buf);
        cptr = get_token(cptr, buf);
        offset = cnv_hex(buf);

        temp = ((unsigned long)segment<<16) + (unsigned long)(offset & 0xFFFF);
        ptr = (BYTE far *)temp;

        dprint("\n%P>", ptr);
        get_string(buf, 50);
        while (buf[0] != NULL_CHAR)
        {
            val = cnv_hex(buf);
            (*ptr) = val;
            ptr++;
```

255

```c
            dprint("\n%P>", ptr);
            get_string(buf, 50);
        }

}   /* End of edit() */


/****************************************************************************
 *
 *   msu_cmd()
 *
 ****************************************************************************/

#define READ        1
#define WRITE           2
#define DUMP        3
#define NONE        0
#define FLASH       1
#define SRAM        2
#define STR_LEN         100

void msu_cmd(char *cptr, int device)
{
    char                buf[10], addr_buf[10];
    unsigned long int   addr;
    int                 type = NONE;
    unsigned int        data;
    int                 action = NONE;
    int                 i = 0;
    int                 j;
    WORD            x;
    static BYTE         str[STR_LEN];
    BYTE            dbuf[256], val;
    static DWORD    daddr = 0;


    cptr = get_token(cptr, buf);

    if (buf[0] == '0')                      /* Turn OFF */
        msu_off(device);

    else if (buf[0] == '1')                 /* Turn ON */
        msu_on(device);

    else
    {
        cptr = get_token(cptr, addr_buf);
        addr = cnv_lhex(addr_buf);

        if (stricmp(buf, "rs") == 0)        /* Read Static */
        {
            if (debug)
                dprint("Reading SRAM @ %lx\n", addr);
            type = SRAM;
            action = READ;
        }

        else if (stricmp(buf, "rf") == 0)  /* Read Flash */
        {
            if (debug)
                dprint("Reading FLASH @ %lx\n", addr);
            type = FLASH;
            action = READ;
        }

        else if (stricmp(buf, "df") == 0)  /* Dump Flash */
        {
            type = FLASH;
            action = DUMP;
            if (addr_buf[0] != '\0')
                daddr = addr;
            /* else, get the next paragraph from the last dump */
        }

        else if (stricmp(buf, "ds") == 0)  /* Dump Static */
        {
            type = SRAM;
            action = DUMP;
            if (addr_buf[0] != '\0')
                daddr = addr;
            /* else, get the next paragraph from the last dump */
        }

        else if (stricmp(buf, "ws") == 0) /* Write Static */
```

```c
{
    if (debug)
        dprint("Writing SRAM @ %lx\n", addr);
    type = SRAM;
    action = WRITE;
}

else if (stricmp(buf, "wf") == 0)  /* Write Flash */
{
    if (debug)
        dprint("Writing FLASH @ %lx\n", addr);
    type = FLASH;
    action = WRITE;
}

else if (stricmp(buf, "ef") == 0)  /* Erase Flash */
{
    if (debug)
        dprint("Erasing FLASH for MSA\n");
    msu_flash_erase(device);
    return;
}

else if (stricmp(buf, "af") == 0)  /* Address Flash */
{
    if (debug)
        dprint("Setting Flash address = %lx\n", addr);
    msu_set_faddr(device, addr);
    return;
}

else if (stricmp(buf, "cf") == 0)  /* Address Flash */
{
    if (debug)
        dprint("Reading Flash Codes....");

    dprint("Flash Codes = %X\n", msu_flash_codes(device));
    return;
}

else if (stricmp(buf, "tf") == 0)  /* Test Flash */
{
    if (debug)
        dprint("Performing Flash test....");

    msu_ftest(device);
    return;
}

else if (stricmp(buf, "ts") == 0)  /* Test SRAM */
{
    msu_stest(device);
}

switch(action)
{
    case READ:
        if (type == SRAM)
            data = msu_sram_read1(device, addr++);
        else
            data = msu_flash_read1(device, addr++);

        while ((data != NULL_CHAR) && (i < STR_LEN-1))
        {
            str[i++] = data;
            if (type == SRAM)
                data = msu_sram_read1(device, addr++);
            else
                data = msu_flash_read1(device, addr++);
        }
        str[i] = NULL_CHAR;
        dprint("%s", str);
        break;

    case DUMP:
        if (type == SRAM)
            msu_sram_read(device, daddr, (BYTE *)&dbuf, 256);
        else
            msu_flash_read(device, daddr, (BYTE *)&dbuf, 256);

        for (i = 0; i < 256; i += 16, daddr += 16)
        {
            if (daddr < 0x10)
                dprint("0");
```

```
                              if (daddr < 0x100)
                                  dprint("0");
                              if (daddr < 0x1000)
                                  dprint("0");
                              if (daddr < 0x10000)
                                  dprint("0");
                              if (daddr < 0x100000)
                                  dprint("0");
                                  dprint("%1X   ", daddr);
                                  for (j = 0; j < 16; j++)
                              {
                                      if (j == 8)
                                          dprint(" ");
                                      val = dbuf[i+j];
                                      if (val < 0x10)
                                          dprint("0");
                                      dprint("%X ", val);
                              }
                                  dprint("   ");
                                  for (j = 0; j < 16; j++)
                              {
                                      val = dbuf[i+j];
                                      if ((val >= 32) && (val <= 127))
                                          dprint("%c", dbuf[i+j]);
                                  else
                                          dprint(".");
                              }
                                  dprint("\n");
                      }

                          break;

                  case WRITE:
                      while ((*cptr != NULL_CHAR) && (i < STR_LEN))
                      {
                          if (type == SRAM)
                              msu_sram_write1(device, addr++, *cptr);
                          else
                              msu_flash_write1(device, addr++, *cptr);
                          i++;
                          cptr++;
                      }
                      if (type == SRAM)
                          msu_sram_write1(device, addr, *cptr);
                      else
                          msu_flash_write1(device, addr, *cptr);
                      break;

                  default:
                      dprint("MSU command error.");

          }   /* End of SWITCH */

      }   /* End of ELSE */

}   /* End of msu_cmd() */


/*****************************************************************************
 *
 *   msa_cmd()
 *
 *****************************************************************************/

void msa_cmd(char *cptr)
{
    msu_cmd(cptr, MSA0);

}   /* End of msa_cmd() */


/*****************************************************************************
 *
 *   msb_cmd()
 *
 *****************************************************************************/

void msb_cmd(char *cptr)
{
    msu_cmd(cptr, MSB0);

}   /* End of msb_cmd() */
```

```
/*****************************************************************************
 *
 *   debug_cmd()
 *
 ****************************************************************************/

void debug_cmd(char *cptr)
{
    char            buf[10];


    cptr = get_token(cptr, buf);

    if (buf[0] == '0')
        debug = FALSE;

    if (buf[0] == '1')
        debug = TRUE;

}   /* End of debug_cmd() */



/*****************************************************************************
 *
 *   time_cmd()
 *
 ****************************************************************************/

void time_cmd(char *cptr)
{
    static int       mdays[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    static char      *dow_str[] = {"Thu", "Fri", "Sat", "Sun", "Mon", "Tue", "Wed"};
    static char      *dom_str[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
                                   "Aug", "Sep", "Oct", "Nov", "Dec"};
    unsigned long t, et, tdays, tsecs;
    char          buf[20];
    int              year, month, day, hour, min, sec, dow, leap;


    cptr = get_token(cptr, buf);

    if (buf[0] == NULL_CHAR)
    {
        t = get_time();
        et = get_elapsed_time();
        tsecs = t;
        tdays = t/SECS_PER_DAY;

        /* first, get elapsed years */
        leap = 2;          /* corresponds with Modulo four, starting with 1970 */
        year = 0;
        while (tdays >= 366)
        {
            if (leap%4 == 0)
            {
                tdays -= 366;
                tsecs -= 366*SECS_PER_DAY;
            }
            else
            {
                tdays -= 365;
                tsecs -= 365*SECS_PER_DAY;
            }

            year++;
            leap++;
        }
        if ((tdays == 365) && !(leap%4 == 0))
        {
            tdays -= 365;
            tsecs -= 365*SECS_PER_DAY;
            year++;
        }

        /* now, get the month */
        month = 1;
        while (tdays >= mdays[month])
        {
            if ((month == 2) && (leap%4 == 0))
            {
                tdays -= 29;
                tsecs -= 29*SECS_PER_DAY;
            }
            else
```

259

```
                {
                    tdays -= mdays[month];
                    tsecs -= mdays[month]*SECS_PER_DAY;
                }
            month++;
        }

        day = tdays;
        tsecs -= day*SECS_PER_DAY;

        if (debug)
        {
            dprint("year=%d, month=%d, day=%d  ", year, month, day);
            dprint("tsecs remaining=%ld\n", tsecs);
        }

        hour = tsecs/SECS_PER_HOUR;
        tsecs -= hour*SECS_PER_HOUR;

        min = tsecs/SECS_PER_MIN;
        tsecs -= min*SECS_PER_MIN;

        sec = tsecs;

        dow = (t/SECS_PER_DAY)%7;                /* 0=Thu, 1=Fri, etc. */

        dprint("%s %d %s %d, ", dow_str[dow], day+1, dom_str[month-1], 1970+year);
        if (hour < 10)
            dprint("0%d:", hour);
        else
            dprint("%d:", hour);
        if (min < 10)
            dprint("0%d:", min);
        else
            dprint("%d:", min);
        if (sec < 10)
            dprint("0%d", sec);
        else
            dprint("%d", sec);

        /* Elapsed time */
        day = et/SECS_PER_DAY;
        et -= day*SECS_PER_DAY;
        hour = et/SECS_PER_HOUR;
        et -= hour*SECS_PER_HOUR;
        min = et/SECS_PER_MIN;
        et -= min*SECS_PER_MIN;
        sec = et;
        dprint("  (Elapsed time = ");
        if (day < 10)
            dprint("0%d:", day);
        else
            dprint("%d:", day);
        if (hour < 10)
            dprint("0%d:", hour);
        else
            dprint("%d:", hour);
        if (min < 10)
            dprint("0%d:", min);
        else
            dprint("%d:", min);
        if (sec < 10)
            dprint("0%d)\n", sec);
        else
            dprint("%d)\n", sec);
    }

else
{
    buf[2] = NULL_CHAR;
    year = atoi(buf) - 70;
    if (year < 0)
        year += 100;      /* compensate for next century */

    buf[5] = NULL_CHAR;
    month = atoi(buf+3) - 1;

    buf[8] = NULL_CHAR;
    day = atoi(buf+6) - 1;

    buf[11] = NULL_CHAR;
    hour = atoi(buf+9);

    buf[14] = NULL_CHAR;
```

```c
        min = atoi(buf+12);

        sec = atoi(buf+15);

        leap = (year+2)/4;          /* number of leap years since 1970 */

        t = year*SECS_PER_YEAR + leap*SECS_PER_DAY;
        while (month > 0)
            t += mdays[month--]*SECS_PER_DAY;

        t += day*SECS_PER_DAY;

        t += (hour*SECS_PER_HOUR) + (min*SECS_PER_MIN) + sec;

        set_time(t);
    }

}   /* End of time_cmd() */


/***************************************************************************
 *
 *   rf_cmd()
 *
 ***************************************************************************/

#define  T0CON     0xFF56
#define  T0CNT     0xFF50
#define  T0CMPA    0xFF52
#define  T0CMPB    0xFF54

#define  T1CON     0xFF5E
#define  T1CNT     0xFF58
#define  T1CMPA    0xFF5A
#define  T1CMPB    0xFF5C

void rf_cmd(char *cptr)
{
    static   BYTE      bits = 0x00;
    char               buf[10];
    int                temp;


    cptr = get_token(cptr, buf);

    if (stricmp(buf, "0") == 0)
        eps_set_power(RF, OFF);

    else if (stricmp(buf, "1") == 0)
        eps_set_power(RF, ON);


    else if (stricmp(buf, "t") == 0)
    {
        bits |= 0x01;
        pcb_write(0, 0, bits);
    }

    else if (stricmp(buf, "r") == 0)
    {
        bits &= ~0x01;
        pcb_write(0, 0, bits);
    }


    else if (stricmp(buf, "tx") == 0)
    {
        bits |= 0x02;
        pcb_write(0, 0, bits);
    }

    else if (stricmp(buf, "rx") == 0)
    {
        bits &= ~0x02;
        pcb_write(0, 0, bits);
    }


    else if (stricmp(buf, "lop") == 0)
    {
        bits &= ~0x04;
        pcb_write(0, 0, bits);
    }
```

261

```c
        else if (stricmp(buf, "loa") == 0)
        {
            bits |= 0x04;
            pcb_write(0, 0, bits);
        }


        else if (stricmp(buf, "lhp") == 0)
        {
            bits &= ~0x08;
            pcb_write(0, 0, bits);
        }

        else if (stricmp(buf, "lha") == 0)
        {
            bits |= 0x08;
            pcb_write(0, 0, bits);
        }


        else if (stricmp(buf, "p") == 0)
        {
            cptr = get_token(cptr, buf);
            temp = cnv_hex(buf);
            temp &= 0x03;
            temp <<= 4;
            bits |= (BYTE)temp;
            pcb_write(0, 0, bits);
        }


        else if (stricmp(buf, "lna0") == 0)
        {
            bits |= 0x40;                       /* reverse logic than all others */
            pcb_write(0, 0, bits);
        }

        else if (stricmp(buf, "lna1") == 0)
        {
            bits &= ~0x40;                      /* reverse logic than all others */
            pcb_write(0, 0, bits);
        }


        else if (stricmp(buf, "hpa0") == 0)
        {
            bits &= ~0x80;
            pcb_write(0, 0, bits);
        }

        else if (stricmp(buf, "hpa1") == 0)
        {
            bits |= 0x80;
            pcb_write(0, 0, bits);
        }


        else if (stricmp(buf, "e") == 0)
        {
            outpw(T0CNT, 0);
            outpw(T0CMPA, 0);               /* maximum count (65536) */
            outpw(T0CON, 0xC001);           /* internal clk, retrigger, CMPA only */

            outpw(T1CNT, 0);
            outpw(T1CMPA, 1);               /* smallest compare A */

            cptr = get_token(cptr, buf);
            if (buf[0] == NULL_CHAR)
            {
                /* default to 5 second RF Enable */
                temp = 140;             /* ~5 seconds */
            }
            else
            {
                temp = cnv_hex(buf);
                temp *= 28;             /* ~28 CMPB per second */
            }

            outpw(T1CMPB, temp);
            outpw(T1CON, 0xC006);       /* ext. clk, one shot, CMPA/CMPB dual mode */
        }

} /* End of rf_cmd() */
```

```
/****************************************************************************
 *
 *    test_cmd()
 *
 ****************************************************************************/

 void    test_cmd(char *cptr)
{
     int       c, x;
     WORD value;
     double    d;

#define  AD_WAIT  5000

     pcb_write(EPS0, 3, 0x70);
     pcb_write(EPS0, 1, 0x50);


     /* Setup MUXes A/B for first temperature sensors (Calibration resistors) */
     pcb_write(TMUXA0, 0, 0x10);
     pcb_write(TMUXB0, 0, 0x10);


     pcb_write(EPS0, 3, 0x80);          /* EPS Port 3 */
     pcb_write(EPS0, 1, 0x30);          /* EPS Port 1 */

     eps_set_port2(eps_get_port2());

     outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
     /* Wait for RESET bit to clear */
     for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0002); x++)
          ;
     if (x == AD_WAIT)
     {
          ad_flag = 1;
          return;
     }


     outpw(AD_CONFIG, 0x0008);          /* Full Calibration */
     /* Wait for CALIBRATION bit to clear */
     for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0008); x++)
          ;
     if (x == AD_WAIT)
     {
          ad_flag = 1;
          return;
     }

     outpw(AD_CONFIG, 0x0000);             /* stop the sequencer and point to RAM 00 */

     /* Program A/D Sequencer: based on schedule for period 0 */
     outpw(AD_INSTR0, 0xF200);
     outpw(AD_INSTR1, 0xF204);
     outpw(AD_INSTR2, 0xF208);
     outpw(AD_INSTR3, 0xF210);
     outpw(AD_INSTR4, 0xF218);
     outpw(AD_INSTR5, 0xF202);          /* Pause */


     /* Setup A/D Timer */
     outpw(AD_TIMER, 2000);

     outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
     /* Wait for RESET bit to clear */
     for (x = 0; (x < AD_WAIT) && (inpw(AD_CONFIG) & 0x0002); x++)
          ;
     if (x == AD_WAIT)
     {
          ad_flag = 1;
          return;
     }

     /* Force A/D to interrupt when 5 readings in the FIFO occur */
/*   outpw(AD_IER, 0x2804); */

     /* Clear any interrupts of the A/D by reading the status register */
/*   inpw(AD_ISR); */

     /* Start the A/D Sequencer.  Interrupt will occur eventually */
     outpw(AD_CONFIG, 0x0001);     /* start sequencer */
```

263

```c
        /* Wait for 5 samples in the FIFO */
        while (((inpw(AD_ISR) & 0xF800) >> 11) < 5)
                ;

        c = (inpw(AD_ISR) & 0xF800) >> 11;
        while (c)
        {
            value = inpw(AD_FIFO);
            dprint("%u) ", (value & 0xE000) >> 13);
            if (value & 0x1000)
                dprint("-");
            dprint("%u, ", (value & 0x0FFF));

            dprint("0x%X, ", (value & 0x0FFF));

            d = (double)(value & 0x0FFF);
            d = (d/4095.0)*5.0;

            dprint("%3.3lf Volts", d);

            switch((value&0xE000)>>13)
            {
                case 0:  /* DCS Temp */
                    d = (d - 0.5)*100;
                    dprint(", DCS Temp. = %3.3lf C", d);
                    break;

                case 1:  /* Modem Temp */
                    d = (d - 0.5)/.01;
                    dprint(", Modem Temp. = %3.3lf C", d);
                    break;

                case 3:  /* TMUXA Temp */
                    dprint(", TMUXA Temp. = %d C", cnv_therm(value&0x0FFF));
                    break;

                case 4:  /* TMUXB Temp */
                    dprint(", TMUXB Temp. = %d C", cnv_therm(value&0x0FFF));
                    break;

                case 2:  /* EPS Measurement */
                    dprint(", EPS");
                    break;

            }

            dprint("\n");
            c--;
        }
}  /* End of test_cmd() */


/******************************************************************************
 *
 *   tx_cmd()
 *
 ******************************************************************************/

void tx_cmd(char *cptr)
{
    int     i;


    i = 0;
    while ((*cptr != NULL_CHAR) && (i < 128))
    {
        cha_out_buf0[i] = *cptr;
        cptr++;
        i++;
    }

    /* Setup the pointer to the source for DMA channel 1 (Tx) */
    _asm
    {
        mov ax, SEG cha_out_buf0
        rol ax, 4
        mov bx, ax
        and ax, 0xFFF0
        add ax, OFFSET cha_out_buf0
        adc bx, 0
        and bx, 0x000F
        mov dx, D1SRCL
        out dx, ax
```

264

```
            mov ax, bx
            mov dx, D1SRCH
            out dx, ax
    }


    /* Setup the pointer to the destination for DMA channel 1 (Tx) */
    outpw(D1DSTH, 0x0000);
    outpw(D1DSTL, SCCA_DATA);

    outpw(D1TC, i);              /* DMA Transfer Count */

    outpw(D1CON, 0x1786);        /* DST: i/o, no inc., no dec.; SRC: mem, inc., no dec.
                                  *      terminate on TC; INT on TC; Dest. Synch;
                                  *      low priority; do not use Tmr2; Byte xfer
                                  *      Start the DMA channel */



}   /* End of tx_cmd() */


/*******************************************************************************
 *
 *  rx_cmd()
 *
 ******************************************************************************/

void rx_cmd(char *cptr)
{
    char buf[10];


    cptr = get_token(cptr, buf);
    if (stricmp(buf, "i") == 0)
    {
        inp(SCCA_DATA);
        inp(SCCA_DATA);
        inp(SCCA_DATA);

        /* Setup the pointer to the destination for DMA channel 0 (Rx) */
        _asm
        {
            mov ax, SEG cha_in_buf0
            rol ax, 4
            mov bx, ax
            and ax, 0xFFF0
            add ax, OFFSET cha_in_buf0
            adc bx, 0
            and bx, 0x000F
            mov dx, D0DSTL
            out dx, ax
            mov ax, bx
            mov dx, D0DSTH
            out dx, ax
        }

        /* Setup the pointer to the source for DMA channel 0 (Rx) */
        outpw(D0SRCH, 0x0000);
        outpw(D0SRCL, SCCA_DATA);

        outpw(D0TC, 514);        /* DMA Transfer Count - include the two CRC bytes */

        outpw(D0CON, 0xA366);    /* DST: mem, inc., no dec.; SRC: i/o, no inc., no dec.
                                  *      terminate on TC; INT on TC; Source Synch;
                                  *      high priority; do not use Tmr2; Byte xfer
                                  *      START the channel */

        dprint("Rx initialized and ready.\n");
        return;
    }


    if (inp(SCCA_CMD) & 0x010)
        dprint("Sync (Flag) not detected, still in Hunt Mode\n");
    else
        dprint("Sync (Flag) detected!\n");


    if (stricmp(buf, "s") == 0)
    {
        dprint("Tx Underrun/EOM: %u, Rx Overrun: %u\n", a_txunderrun_eom, a_rxoverrun);
        dprint("Break/Abort: %u\n", a_brk_abort);
    }
```

265

```
        if (rx_eom)
        {
            dprint("%s\n", cha_in_buf0);
            rx_eom = FALSE;
        }
        else
            dprint("No message received.\n");

}   /* End of rx_cmd() */


/*****************************************************************************
 *
 *  eps_cmd()
 *
 *****************************************************************************/

void eps_cmd(char *cptr)
{
    char    buf[10];
    static char  *ctrls[]={"HeatA", "", "TMUXA", "MSA", "", "", "", "",
                    "", "", "Ant-Rel", "MSB", "TMUXB", "RF", "HeatB", ""};
    static char  *bat_sw[]={"B Trickle", "B Online", "B Discharge", "B Charge",
                    "A Trickle", "A Online", "A Discharge", "A Charge"};
    static int   bat_cmd1[2][9] = {{0, 0, 0, 0, 0, 1, 3, 5, 7}, {0, 0, 0, 0, 0, 9, 11, 13, 15} };
    static int   bat_cmd2[2][9] = {{0x70, 0x90, 0xF1, 0xF3, 0xF5, 0x10, 0x10, 0x10, 0x10},
                        {0xB0, 0xD0, 0xF7, 0xF9, 0xFB, 0x10, 0x10, 0x10, 0x10}};
    static double bat_cnv[2][9] = {{1, 1, .585, .585, .585, .409, .409, .409, .409},
                        {1, 1, .525, .525, .525, .21, .21, .21, .21}};
    static int   i_sp_cmd[8] = {0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70};
    static int   i_sp_tbl[8] = {4, 5, 7, 9, 11, 13, 14, 16};
    int     temp, n;
    int     bat, dev;
    int     x;
    WORD a, tempw;
    double  d, sign;
    BYTE b;


    cptr = get_token(cptr, buf);
    switch(buf[0])
    {
        case 'b':
        case 'B':
            cptr = get_token(cptr, buf);
            if (stricmp(buf, "a") == 0)
                bat = BAT_A;
            else if (stricmp(buf, "b") == 0)
                bat = BAT_B;
            else if (buf[0] == NULL_CHAR)
            {
                tempw = eps_get_battery();
                dprint("Battery Controls that are ON (0x%X):\n", tempw);
                for (x = 0; x <= 15; x++)
                    if (tempw & (1<<x))
                        dprint("  %s", bat_sw[x]);
                dprint("\n");
                return;
            }
            else
            {
                dprint("EPS cmd error: battery (%s) not recognized.\n", buf);
                return;
            }

            cptr = get_token(cptr, buf);
            if (stricmp(buf, "c") == 0)
                temp = BAT_CHARGE_ON;
            else if (stricmp(buf, "d") == 0)
                temp = BAT_DISCHARGE_ON;
            else if (stricmp(buf, "o") == 0)
                temp = BAT_ONLINE;
            else if (stricmp(buf, "t") == 0)
                temp = BAT_TRICKLE_ON;
            else
            {
                dprint("EPS cmd error: control (%s) not recognized.\n", buf);
                return;
            }


            cptr = get_token(cptr, buf);
            if (stricmp(buf, "on") == 0)
```

```
            eps_set_battery(bat, temp);
        else if (stricmp(buf, "off") == 0)
            eps_set_battery(bat, temp+1);
        else
            dprint("EPS cmd error: option (%s) error.: %s\n", buf);
        break;


    case 'c':
    case 'C':
        cptr = get_token(cptr, buf);
        if (stricmp(buf, "tmuxa") == 0)
            dev = PWR_TMUXA;
        else if (stricmp(buf, "tmuxb") == 0)
            dev = PWR_TMUXB;
        else if (stricmp(buf, "msa") == 0)
            dev = PWR_MSA;
        else if (stricmp(buf, "msb") == 0)
            dev = PWR_MSB;
        else if (stricmp(buf, "heata") == 0)
            dev = PWR_HEATA;
        else if (stricmp(buf, "heatb") == 0)
            dev = PWR_HEATB;
        else if (stricmp(buf, "rf") == 0)
            dev = PWR_RF;
        else if (stricmp(buf, "antrel") == 0)
            dev = PWR_ANTREL;
        else if (buf[0] == NULL_CHAR)
        {
            tempw = eps_get_power();
            dprint("Subsystems that are ON (0x%X):\n", tempw);
            for (x = 0; x <= 15; x++)
                if (tempw & (1<<x))
                    dprint("  %s", ctrls[x]);
            dprint("\n");
            return;
        }
        else
        {
            dprint("EPS cmd error: subsystem (%s) not recognized.\n", buf);
            return;
        }

        cptr = get_token(cptr, buf);
        if (stricmp(buf, "on") == 0)
            eps_set_power(dev, ON);
        else if (stricmp(buf, "off") == 0)
            eps_set_power(dev, OFF);
        else
            dprint("EPS cmd error: power control syntax error: %s.\n", buf);
        break;


    case 'v':
    case 'V':
        cptr = get_token(cptr, buf);
        if ((buf[0] == 'A') || (buf[0] == 'a'))
        {
            bat = BAT_A;
            n = atoi(&(buf+1));
        }
        else if ((buf[0] == 'B') || (buf[0] == 'b'))
        {
            bat = BAT_B;
            n = atoi(&(buf+1));
        }
        else if ((buf[0] == 'S') || (buf[0] == 's'))
        {
            bat = BAT_NONE;
            n = 0;
        }
        else if ((buf[0] == NULL_CHAR) || (temp < 0) || (temp > 8))
        {
            dprint("EPS cmd error: voltage source (%s) not recognized.\n", buf);
            return;
        }


        /* Setup the EPS MUXes via the PCB */
        if (bat != BAT_NONE)
        {
            if ((n >= 5))
                pcb_write(EPS0, 3, bat_cmd1[bat][n]);
            pcb_write(EPS0, 1, bat_cmd2[bat][n]);
```

267

```
            dprint("Battery ");
            if (bat == BAT_A)
                dprint("A");
            else
                dprint("B");
            dprint(" Cell # %d (accum.) ", n);

            a = adr(4);        /* EPS uses channel 4 of the A/D */
            d = a;
            d = 5.0*(d/4095);

            dprint("%u (0x%X), %3.3lf V -> %3.3lf V\n", a, a, d, d/bat_cnv[bat][n]);
        }

    else/* its the s/c bus */
    {
            dprint("S/C ");
            pcb_write(EPS0, 1, 0xFD);
            a = adr(4);        /* EPS uses channel 4 of the A/D */
            d = a;
            d = 5.0*(d/4095);

            dprint("%u (0x%X), %3.3lf V -> %3.3lf V\n", a, a, d, d*3.41);
    }

    break;


case 'i':
case 'I':
    cptr = get_token(cptr, buf);
    if ((buf[0] == 'A') || (buf[0] == 'a'))
        bat = BAT_A;
    else if ((buf[0] == 'B') || (buf[0] == 'b'))
        bat = BAT_B;
    else if ((buf[0] == 'S') || (buf[0] == 's'))
    {
        bat = BAT_NONE;
        n = -1;
    }
    else if ((buf[0] == 'P') || (buf[0] == 'p'))
    {
        bat = BAT_NONE;
        n = atoi(&(buf+1));
    }

    else if (buf[0] == NULL_CHAR)
    {
        dprint("EPS cmd error: current (%s) not recognized.\n", buf);
        return;
    }

    /* Setup the EPS MUXes via the PCB */
    if (bat == BAT_NONE)
    {
        if (n == -1) /* its the s/c bus */
        {
            pcb_write(EPS0, 3, 0x80);
            b = eps_get_port2() | (BYTE)0x01; eps_set_port2(b);
            pcb_write(EPS0, 1, 0x30);
            a= adr(4);
            d = a;
            dprint("S/C current = %u (0x%X), %.3lf V -> %.3lf mA\n",
                    a , a, d/819.0, 1000.0*((d*0.002442)-5.0));
        }
        else
        {
            pcb_write(EPS0, 3, i_sp_cmd[n]);
            b = eps_get_port2() | (BYTE)0x01; eps_set_port2(b);
            pcb_write(EPS0, 1, 0x50);
            a = adr(4);
            d = a;
            dprint("S/P:%d (#%d) current = %u (0x%X), %.3lf V -> %.3lf mA\n",
                    n, i_sp_tbl[n], a, a, d/819.0, 1000.0*(d*0.000488 - 1.0) );
        }
    }
    else      /* one of the batteries */
    {
        dprint("Battery ");
        if (bat == BAT_A)
        {
            dprint("A");
            pcb_write(EPS0, 3, 0x90);
```

```
            }
            else
            {
                dprint("B");
                pcb_write(EPS0, 3, 0xA0);
            }
            b = eps_get_port2() | (BYTE)0x01; eps_set_port2(b);
            pcb_write(EPS0, 1, 0x30);
            a = adr(4);
            d = a;

            /* Read the direction. */
            n = pcb_read(EPS1, 1);    /* Port 5 of the EPS */
            if (bat == BAT_A)
                temp = 0x01;
            else
                temp = 0x02;
            sign = (temp & n) ? 1.0 : -1.0;
            dprint(" current = %u (0x%X), %3lf V ->%3lf mA\n",
                        a, a, sign*(d/819.0), 1000.0*sign*((d*0.002442)-5.0));
        }

        break;


    case 'w':
    case 'W':
        eps_reset_wdog();
        dprint("Watchdog timer reset.\n");
        break;


    default:
        dprint("EPS cmd error.\n");
        break;

    } /* End of SWITCH */

} /* End of eps_cmd() */



/*****************************************************************************
 *
 *  WORD adr()
 *
 *****************************************************************************/

WORD adr__old(int ch)
{
    unsigned int c, value, isr, temp, value_save;
    double          x;


    outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)  /* Wait for Reset bit to clear */
        ;

    outpw(AD_CONFIG, 0x0008);          /* Full Calibration */
    while (inpw(AD_CONFIG) & 0x0008)  /* Wait for calibration to finish */
        ;

    outpw(AD_CONFIG, 0x0000);          /* Stop sequencer, point to RAM 00 */


    /* DCS Temperature, MUX+ = IN0, MUX- = GND */
    outpw(AD_INSTR0, 0xF200);      /* acq. time = full way, no wdog, 12-bit, */
                                    /*  Timer ON, NO sync, Vin- =Gnd, Vin+ =IN0 */
                                    /*  Pause = NO, loop = NO */

    /* Modem Temperature, MUX+ = IN1, MUX- = GND */
    outpw(AD_INSTR1, 0xF204);      /* acq. time = full way, no wdog, 12-bit, */
                                    /*  Timer ON, NO sync, Vin- =Gnd, Vin+ =IN1 */
                                    /*  no pause, loop = NO */

    /* TMUXA, MUX+ = IN4, MUX- = GND */
    outpw(AD_INSTR2, 0xF210);      /* acq. time = full way, no wdog, 12-bit, */
                                    /*  Timer ON, NO sync, Vin-= Gnd, Vin+= IN4 */
                                    /*  no pause, loop = NO */

    /* TMUXB, MUX+ = IN6, MUX- = GND */
    outpw(AD_INSTR3, 0xF218);      /* acq. time = full way, no wdog, 12-bit, */
                                    /*  Timer ON, NO sync, Vin- =Gnd, Vin+ =IN6 */
                                    /*  no pause, loop = NO */
```

```c
        /* EPS, MUX+ = IN2, MUX- = GND */
        outpw(AD_INSTR4, 0xF208);        /* acq. time = full way, no wdog, 12-bit, */
                                         /*  Timer ON, NO sync, Vin- =Gnd, Vin+ =IN1 */
                                         /*  no pause, loop = NO */


        /* Dummy instruction to PAUSE */
        outpw(AD_INSTR5, 0xF202);


        /* RAM 01(1) and 10(2) are not set - limit stuff */


        /* Timer to slow the conversion rate */
        outpw(AD_TIMER, 1000);


        outpw(AD_CONFIG, 0x0002);        /* Reset the A/D */
        while (inpw(AD_CONFIG) & 0x0002)  /* Wait for Reset bit to clear */
            ;

        outpw(AD_CONFIG, 0x0001); /* start sequencer */

        /* Wait for INT 5 - Pause Interrupt */
        while (!inpw(AD_ISR & 0x0020))
            ;
        /* Sequencer is stopped, due to Pause in last instruction */

        /* Wait for 5 samples in the FIFO */
        while (((inpw(AD_ISR) & 0xF800) >> 11) < 5)
            ;

        c = (inpw(AD_ISR) & 0xF800) >> 11;
        while (c)
        {
            value = inpw(AD_FIFO);

            if (((value&0xE000)>>13) == ch)
                value_save = value & 0x0FFF;

            c--;
        }

        return(value_save);

}   /* End of adr() */


/*****************************************************************************
 *
 *   tmux_cmd()
 *
 ****************************************************************************/

void tmux_cmd(char *cptr)
{
    char buf[10];
    int     ch;
    WORD a;
    double  x;


    cptr = get_token(cptr, buf);
    switch(buf[0])
    {
        case 'a':
        case 'A':
            cptr = get_token(cptr, buf);
            if (stricmp(buf, "on") == 0)
                eps_set_power(PWR_TMUXA, ON);
            else if (stricmp(buf, "off") == 0)
                eps_set_power(PWR_TMUXA, OFF);
            else
            {
                ch = atoi(buf);
                if ((ch < 0) || (ch > 31))
                {
                    dprint("TMUX: bad cmd (%s)\n", buf);
                    return;
                }

                pcb_write(TMUXA, 0, 0x10 + ch);
                a = adr(2);
                x = a;
```

270

```
                    x = (x/4095.0)*5.0;
                    dprint("TMUXA Channel %d: %u (0x%X), %3.3lf V -> %d C",
                           ch, a & 0x0FFF, a & 0x0FFF, x, cnv_therm(a));
                }

                break;

        case 'b':
        case 'B':
                cptr = get_token(cptr, buf);
                if (stricmp(buf, "on") == 0)
                    eps_set_power(PWR_TMUXB, ON);
                else if (stricmp(buf, "off") == 0)
                    eps_set_power(PWR_TMUXB, OFF);
                else
                {
                    ch = atoi(buf);
                    if ((ch < 0) || (ch > 31))
                    {
                        dprint("TMUX: bad cmd (%s)\n", buf);
                        return;
                    }

                    pcb_write(TMUXB, 0, 0x10 + ch);
                    a = adr(3);
                    x = a;
                    x = (x/4095.0)*5.0;
                    dprint("TMUXB Channel %d: %u (0x%X), %3.3lf V -> %d C",
                           ch, a & 0x0FFF, a & 0x0FFF, x, cnv_therm(a));
                }
                break;

        default:
                dprint("TMUX: bad MUX (%s).\n", buf);
    }

}  /* End of tmux_cmd() */


/*******************************************************************************
 *
 *   tlm_cmd()
 *
 *******************************************************************************/


void tlm_cmd(char *cptr)
{
    int      i;
    BYTE *ptr = (BYTE *)&tlm_record;


    serial_out(CTRL_X);
    for (i = 0; i < sizeof(tlm_record_struct); i++)
        serial_out(*ptr++);

}  /* End of tlm_cmd() */


/*******************************************************************************
 *
 *   read_eps_ad()
 *
 *******************************************************************************/

unsigned int read_eps_ad(void)
{
    outpw(AD_CONFIG, 0x0002);          /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)   /* Wait for Reset bit to clear */
        ;

    outpw(AD_CONFIG, 0x0008);          /* Full Calibration */
    while (inpw(AD_CONFIG) & 0x0008)   /* Wait for calibration to finish */
        ;


    /* EPS, MUX+ = IN2, MUX- = GND */
    /* outpw(AD_INSTR0, 0xF269); */          /* Vin- = IN3, Vin+ = IN2 */
    outpw(AD_INSTR0, 0xF209);      /* acq. time = 1/2 way, no wdog, 12-bit, */
                                        /* Timer ON, NO sync, Vin- =Gnd, Vin+ =IN2 */
                                        /* no pause, loop = YES */

    /* Timer to slow the delay before acquisition and consequent conversion */
    outpw(AD_TIMER, 1000);             /* 32 clocks * 1000 = ~ 4 msec */
```

271

```c
        outpw(AD_CONFIG, 0x0002);           /* Reset the A/D */
        while (inpw(AD_CONFIG) & 0x0002)    /* Wait for Reset bit to clear */
            ;

        outpw(AD_CONFIG, 0x0001); /* start sequencer */

        /* Wait for INT 5 - Pause Interrupt */
        while (!inpw(AD_ISR & 0x0020))
            ;
        /* Sequencer is stopped, due to Pause in last instruction */

        /* Wait for 1 sample in the FIFO */
        while (((inpw(AD_ISR) & 0xF800) >> 11) < 1)
            ;

        return(inpw(AD_FIFO) & 0x0FFF);

}   /* End of read_eps_ad() */


/*********************************************************************************
 *
 *   adr()
 *
 *********************************************************************************/

unsigned int adr(int ch)
{
    int       n, i;
    unsigned int total;


    outpw(AD_CONFIG, 0x0002);           /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)    /* Wait for Reset bit to clear */
        ;

    outpw(AD_CONFIG, 0x0008);           /* Full Calibration */
    while (inpw(AD_CONFIG) & 0x0008)    /* Wait for calibration to finish */
        ;
    outpw(AD_CONFIG, 0x0000);           /* Stop sequencer, point to RAM 00 */

    /* Loop Bits set for all */
    n = 1;
    switch(ch)
    {
        case 0:        /* DCS: Single-ended, Vin+ = IN0 */
            outpw(AD_INSTR0, 0xF200);
            outpw(AD_INSTR1, 0xF202);
            break;

        case 1:        /* MODEM: Single-ended, Vin+ = IN1 */
            outpw(AD_INSTR0, 0xF204);
            outpw(AD_INSTR1, 0xF202);
            break;

        case 2:        /* TMUXA: Single-ended, Vin+ = IN4 */
            outpw(AD_INSTR0, 0xF210);
            outpw(AD_INSTR1, 0xF202);
            break;

        case 3:        /* TMUXB: Single-ended, Vin+ = IN6 */
            outpw(AD_INSTR0, 0xF218);
            outpw(AD_INSTR1, 0xF202);
            break;

        case 4:        /* EPS: Single-ended, Vin+ = IN2 */
            outpw(AD_INSTR0, 0xF208);
            outpw(AD_INSTR1, 0xF202);
            break;

        default:
            return(0);
            break;

    }   /* End of SWITCH */


    /* Timer to slow the delay before acquisition and consequent conversion */
    outpw(AD_TIMER, 1000);                  /* 32 clocks * 1000 = ~ 4 msec */

    outpw(AD_CONFIG, 0x0002);           /* Reset the A/D */
    while (inpw(AD_CONFIG) & 0x0002)    /* Wait for Reset bit to clear */
        ;
```

272

```c
        outpw(AD_CONFIG, 0x0001); /* start sequencer */

        /* Wait for INT 5 - Pause Interrupt */
        while (!inpw(AD_ISR & 0x0020))
                ;
        /* Sequencer is stopped, due to Pause in last instruction */

        /* Wait for n sample(s) in the FIFO */
        while (((inpw(AD_ISR) & 0xF800) >> 11) < n)
                ;

        if (n == 1)
                return(inpw(AD_FIFO) & 0x0FFF);

        else
        {
                for (total = 0, i = 0; i < n; i++)
                        total += inpw(AD_FIFO) & 0x0FFF;
                return(total/n);
        }

}   /* End of adr() */


/******************************************************************************
 *
 *   sbutdown_cmd()
 *
 ******************************************************************************/

void shutdown_cmd(char *cptr)
{
        pcb_write(EPS0, 0, 0);      /* Battery A control, TMUXA, HEATA */
        pcb_write(EPS0, 2, 0);      /* Other subsystem power */
        pcb_write(EPS1, 2, 0);      /* Battery B control */

        serial_out(CTRL_V);

        while (1)
                ;

}   /* End of shutdown_cmd() */


/******************************************************************************
 *
 *   bcm_cmd()
 *
 ******************************************************************************/

void bcm_cmd(char *cptr)
{
        char buf[10];
        int     ch;
        WORD a;
        double  x;


        cptr = get_token(cptr, buf);

        if (stricmp(buf, "on") == 0)
                bcm_on = TRUE;
        else if (stricmp(buf, "off") == 0)
                bcm_on = FALSE;
        else
        {
                dprint("Battery Charge Monitor is ");
                if (bcm_on)
                        dprint("ON.\n");
                else
                        dprint("OFF.\n");
        }

}   /* End of bcm_cmd() */
```

End of stpi.h, stpi.c

terms.h, terms.c

```
/***************************************************************************
 *
 *   TERMS.H
 *
 *   Routines for terminal manipulation.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *
 *   Revision History:
 *   ==================
 *   Date            Who     What
 *   ---------------+-------+-----------------------------------------------
 *   13 August 1991  Jah     Creation
 *   11 Oct 1991     Jah     Flight terminal routines (assume printer)
 *   2 Nov 1993      Jah      Adopted for DCS
 *
 ***************************************************************************/


#ifdef   TERMS
#endif

#ifndef  TERMS
     extern void  clr(void);
     extern void  home(void);
     extern void  mov_xy(int , int);
     extern void  erase_to_eol(void);
     extern void  mov_b(int);
     extern void  mov_f(int);
     extern void  mov_u(int);
     extern void  mov_d(int);
#endif


/***************************************************************************
 *
 *   TERMS.C
 *
 *   Routines for terminal manipulation.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   ==================
 *   Date            Who     What
 *   ---------------+-------+-----------------------------------------------
 *   13 August 1991  Jah     Creation
 *   11 Oct 1991     Jah     Flight terminal routines (assume printer)
 *   2 Nov 1993      Jah            Adopted for DCS
 *
 ***************************************************************************/


#include        "gen_defs.h"

#define         TERMS
#include        "terms.h"
#undef          TERMS

#include        "print.h"
#include        "scc.h"


/***************************************************************************
 *
 *   clr()
 *
 ***************************************************************************/

void    clr(void)
{
```

```
    serial_out(0x1b);    /* ESC  */
    serial_out(0x5b);    /* [    */
    serial_out(0x32);    /* 2    */
    serial_out(0x4a);    /* J    */


}  /* End of clr() */


/***********************************************************************
 *
 *   erase_to_eol()
 *
 ***********************************************************************/

void    erase_to_eol(void)
{
    serial_out(0x0D);
    serial_out(0x1B);
    serial_out('T');

}  /* End of erase_to_eol() */


/***********************************************************************
 *
 *   home()
 *
 ***********************************************************************/

void    home()
{
    serial_out(0x1b);    /* ESC  */
    serial_out(0x5b);    /* [    */
    serial_out(0x3b);    /* ;    */
    serial_out(0x48);    /* H    */

}  /* End of home()


/***********************************************************************
 *
 *    mov_b() - Moves cursor back x characters "ESC[#D"
 *
 ***********************************************************************/

void mov_b(int x)
{
    while(x--)
    serial_out(0x28);

}  /* End of mov_b() */


/***********************************************************************
 *
 *
 *    mov_f() - Moves cursor forward x characters "ESC[#C"
 *
 *
 ***********************************************************************/

void mov_f(int x)
{

    while(x--)
    serial_out(0x2C);

}  /* End of mov_f() */


/***********************************************************************
 *
 *
 *    mov_u() - Moves cursor up x rows "ESC[#A"
 *
 *
 ***********************************************************************/

void mov_u(int x)
{
    while(x--)
    serial_out(0x2B);
```

275

```
}   /* End of mov_u()*/


/**********************************************************************
 *
 *
 *    mov_d() - Moves cursor back x characters "ESC[#B"
 *
 *
 **********************************************************************/

void mov_d(int x)
{
    while(x--)
    serial_out(0x2A);

}   /* End of mov_d() */


/**********************************************************************
 *
 *        mov_xy()
 *
 **********************************************************************/

void    mov_xy(int x, int y)
{
    serial_out(0x1B);
    serial_out('=');
    serial_out((BYTE)(' ' + y - 1));
    serial_out((BYTE)(' ' + x - 1));

}   /* End of mov_xy() */
```

End of terms.h, terms.c

tlm.h, tlm.c

```
/**********************************************************************
 *
 *   TLM.H
 *
 *   Data types and equates for Pansat Telemetry.
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   ==================
 *   who       when       what
 *   -------+-----------+---------------------------------------------------
 *   Jah       9 June 95    Creation (hardware sensors)
 *   Jah       27 June 95   Software sensors
 *   Jah       26 April 96  Adopted for ROM Startup
 *
 **********************************************************************/



#define  NUM_TS             64      /* Number of Thermistors */
#define  NUM_TM             2       /* Number of IC Temperature Sensors */
#define  NUM_BAT_CELLS      9       /* Number of Battery cells (per battery) */
#define  TOP_CELL      8
#define  NUM_SP             8       /* Number of Solar Panel Current Sensors */

/* Jah */
#define  NUM_SETS       14      /* Number of Sets in Sequencer Instruction Table */
#define  CURRENTS_SAMPLED 5     /* Number of currents sampled per complete A/D sweep */

#define  TLM_RECORD_TIME   (TWO_MINUTES)

/**********************************************************************
 *
 *   PANSAT Hardware Sensors for ROM Startup.
 *
 *   This structure can be used both for the raw data read from the
 *   LM12458 A/D converter (12-bit values reflecting a voltage reading),
 *   as well as converted values in the appropriate units needed for
 *   decision making routines.
 *
 **********************************************************************/

typedef struct sensors
{
    /* Thermistors */
    unsigned int ts[NUM_TS];

    /* IC Temperature Sensors */
    unsigned int tmp[NUM_TM];

    /* Voltages */
    unsigned int vbatta[NUM_BAT_CELLS];
    unsigned int vbattb[NUM_BAT_CELLS];
    unsigned int vscbus;

    /* Currents */
    unsigned int ibatta[CURRENTS_SAMPLED]; /* over the 42 periods */
    unsigned int ibattb[CURRENTS_SAMPLED]; /* over the 42 periods */
    unsigned int iscbus[CURRENTS_SAMPLED]; /* over the 42 periods */
    unsigned int isolar[NUM_SP];


} sensors_struct;


typedef struct cnv_sensors
{
    /* Thermistors */
    signed char  ts[NUM_TS];

    /* IC Temperature sensors */
    signed char      tmp[NUM_TM];

    /* Voltages */
    double   vbatta;
    double   vbattb;
    double   vcellsa[NUM_BAT_CELLS];
```

```
        double   vcellsb[NUM_BAT_CELLS];
        double   vcellsa_avg;
        double   vcellsb_avg;
        double   vscbus;

        /* Currents */
        double   ibatta;
        double   ibattb;
        double   iscbus;
        double   isp[NUM_SP];

}   cnv_sensors_struct;


/****************************************************************************
 *
 *   PANSAT Hardware Configuration
 *
 ****************************************************************************/

/* hw_cfg_struct holds software configuration information regarding
 * hardware systems which is to be recorded to the mass storage unit and
 * available for downloading along with recorded hardware sensors.
 */
typedef struct   hw_cfg
{
    unsigned char              epscfg[3];      /* Ports 0, 2, and 6 */
    unsigned char              pmaxcfg[19]; /* Paramax regsiters + h/w status */
    unsigned char              rfcfg;          /* RF configuration */

} hw_cfg_struct;


/* Indexing into pmaxcfg[] */
#define  PM_AGC_STATUS1    0        /* AGC Status */
#define  PM_IPFI_H         1        /* I Prefilter High Byte */
#define  PM_IPFI_L         2        /* I Prefilter Low Byte */
#define  PM_QPFI_H         1        /* Q Prefilter High Byte */
#define  PM_QPFI_L         4        /* Q Prefilter Low Byte */
#define  PM_TM_CMD_0       5        /* Time 32 bit frequency command (LSB) */
#define  PM_TM_CMD_1       6        /* Time 32 bit frequency command */
#define  PM_TM_CMD_2       7        /* Time 32 bit frequency command */
#define  PM_TM_CMD_3       8        /* Time 32 bit frequency command (MSB) */
#define  PM_PH_CMD_0       9        /* Phase 32 bit frequency command (LSB) */
#define  PM_PH_CMD_1       10       /* Phase 32 bit frequency command */
#define  PM_PH_CMD_2       11       /* Phase 32 bit frequency command */
#define  PM_PH_CMD_3       12       /* Phase 32 bit frequency command (MSB) */
#define  PM_PNCA_L         13       /* PN Correlation Detector Accumulator (LSB) */
#define  PM_PNCA_H         14       /* PN Correlation Detector Accumulator (MSB) */
#define  PM_PNCS_L         15       /* PN Correlation Detector Slip Counter (LSB) */
#define  PM_PNCS_H         16       /* PN Correlation Detector Slip Counter (MSB) */
#define  PM_PNG            17       /* PN Generator Status */
#define  PM_HW             18       /* Paramax hardware interface register */

/* Indexing into epscfgp[] */
#define  EPS_PORT0         0
#define  EPS_PORT2         1
#define  EPS_PORT6         2


/****************************************************************************
 *
 *   PANSAT Software Sensors/Configuration/Statistics
 *
 ****************************************************************************/

/* sw_cfg holds all software configuration information which reflects the current
 * state of the satellite.
 */
typedef struct sw_info
{
    long                  tod;        /* Current data: UTC since 1 Jan 1970 */
    int                     ver;        /* OS version */

    unsigned long int     passcount;    /* Password counter */
    unsigned long int     suaccess;     /* # of successful NPS accesses */
    unsigned long int     sufailed;     /* # of failed NPS accesses */

    unsigned long int     softerr;    /* EDAC: Soft Error count */
    long                  serrtod;    /* EDAC: Time of last soft error */
    char far              *serraddr;    /* EDAC: Current RAM Wash address */

} sw_info_struct;
```

278

```c
    hw_cfg_struct      hw_cfg;


/****************************************************************************
 *
 *   Complete Pansat Telemetry (hardware and software)
 *
 ****************************************************************************/

/* For current (most recent, not stored to the mass storage unit) telemetry. */
typedef struct   tlm_recent
{
    sensors_struct    sensors;

    sw_info_struct    sw_info;

    DWORD              etime;        /* elapsed operating time */
    DWORD              tod;    /* time/date */

}  tlm_recent_struct;


/* For stored telemetry (in mass storage unit).
 *
 * The data stream containing stored telemetry is a series of these structures
 * of which the size is given before the data download begins.
 */
typedef struct   tlm_record
{
    sensors_struct         sensors;

    bcm_info_struct        bcm;
    DWORD                  etime;        /* elapsed operating time */
    DWORD                  tod;    /* time/date */

/* Jah */
/*   hw_cfg_struct          hw_cfg; */

    WORD               crc;

}  tlm_record_struct;


#ifdef TLM
    /* Defines for conversion routines */
    #define  THERM_LOW     -31
    #define  THERM_HIGH    88
    #define  THERM_TAB_SIZE ((THERM_HIGH - (THERM_LOW)) + 1)

    void convert_ad(void);
    int     cnv_therm(WORD n);

#endif


#ifndef TLM
    extern void convert_ad(void);
    extern int   cnv_therm(WORD n);

    extern void  check_tlm(void);

    extern cnv_sensors_struct      tlm_cnv;

    /* Storage for the most recent TLM gathering from the A/D.  These are
     * raw data points.
     */
    extern tlm_recent_struct       tlm;

    /* Define storage for the data to be recorded to the mass storage units.
     */
    extern tlm_record_struct       tlm_record;
#endif
```

```
/**************************************************************************
 *
 *   TLM.C
 *
 *   Petite Amateur Navy Satellite (PANSAT).
 *   Embedded ROM software.
 *   Copyright (c) 1996 Space Systems Academic Group, Naval Postgradate School.
 *   Jim A. Horning (Jah)
 *
 *   Revision History:
 *   =================
 *   who      when       what
 *   -------+-----------+----------------------------------------------------
 *   Jah      30 April 96  Creation
 *
 **************************************************************************/


#include "gen_defs.h"

#include "bcm.h"

#define     TLM
#include "tlm.h"
#undef      TLM

#include "ad.h"
#include "clock.h"
#include "pcb.h"


/* Define storage for the most recent TLM gathering from the A/D.  These are
 * raw data points.
 */
tlm_recent_structtlm;

/* Define storage for the data to be recorded to the mass storage units.
 */
tlm_record_structtlm_record;

/* Define storage for converted tlm values for decision making.
 */
cnv_sensors_struct    tlm_cnv;


/* Starting addresses to begin recording telemetry records to Flash */
static DWORD msaf_tlm_ptr = 0;
static DWORD msbf_tlm_ptr = 0;


/* Conversion factors for telemetry sensors into cnv_sensors */
/* Thermistors: This table begins for the temperature beginning
 *    at -31 C, and working up per degree C.  The table is used
 *    in a binary search.  Table entries are used as follows:
 *    If the A/D value is 4000, then the temperature considered
 *    to be -30C, because it is less than the first entry in the
 *    table, 4191, which corresponds to -31C.
 */
static const int cnv_therm_tab[THERM_TAB_SIZE] =
{
    /* -31 -> -22 */
    4191,    3949,    3723,    3512,    3313,    3127,    2952,    2788,    2634,    2490,
    /* -21 -> -12 */
    2354,    2226,    2106,    1993,    1887,    1787,    1693,    1604,    1520,    1442,
    /* -11 -> -2 */
    1368,    1298,    1231,    1169,    1110,    1055,    1002,    953, 906, 862,
    /*   -1 ->   8 */
    820, 780, 743, 707, 673, 642, 611, 583, 556, 530,
    /*   9 ->  18 */
    506, 482, 461, 440, 420, 401, 383, 366, 350, 335,
    /*  19 ->  28 */
    320, 306, 293, 281, 269, 257, 246, 236, 226, 217,
    /*  29 ->  38 */
    208, 199, 191, 183, 176, 169, 162, 156, 150, 144,
    /*  39 ->  48 */
    138, 133, 127, 123, 118, 113, 109, 105, 101, 97,
    /*  49 ->  58 */
    94,      90,      87,      84,      81,      78,      75,      72,      70,      67,
    /*  59 ->  68 */
    65,      62,      60,      58,      56,      54,      52,      51,      49,      47,
    /*  69 ->  78 */
    46,      44,      43,      41,      40,      38,      37,      36,      35,      34,
    /*  79 ->  88 */
    33,      32,      31,      30,      29,      28,      27,      26,      25,      24
```

```c
};

/* Voltages */
static const double  cnv_vbatts[BCM_NUM_BATS][BCM_NUM_CELLS] =
{
    {1.0*AD_RES,       1.0*AD_RES,       1.90476*AD_RES,  1.90476*AD_RES,  1.90476*AD_RES,
     4.76191*AD_RES, 4.76191*AD_RES,  4.76191*AD_RES,  4.76191*AD_RES},
    {1.0*AD_RES,       1.0*AD_RES,       1.90476*AD_RES,  1.90476*AD_RES,  1.90476*AD_RES,
     4.76191*AD_RES, 4.76191*AD_RES,  4.76191*AD_RES,  4.76191*AD_RES}
};

#define  CNV_VSC       (3.41*AD_RES)


/* Currents */
#define  CNV_IBATTS    (2.0*AD_RES)
#define  CNV_ISC       (2.0*AD_RES)
#define  CNV_ISP       (2.0*AD_RES)


/*******************************************************************************
 *
 *   void convert_ad()
 *
 *   Take all A/D raw values stored in the telemetry structure, .sensors
 *   substructure, and convert it to floating point values that are in the
 *   correct units for higher-level control routines.
 *
 *******************************************************************************/

void convert_ad(void)
{
    register int i;
    double   vbatta[NUM_BAT_CELLS];
    double   vbattb[NUM_BAT_CELLS];
    double   ia, ib, isc;


    /* Convert Thermistors */
    for (i = 0; i < NUM_TS; i++)
        tlm_cnv.ts[i] = cnv_therm(tlm.sensors.ts[i]);


    /* Convert Temperature Sensors */
    for (i = 0; i < NUM_TM; i++)
        tlm_cnv.tmp[i] = (tlm.sensors.tmp[i]*AD_RES - 0.5)*100.0;


    /* Convert Voltages: Battery A, Battery B, and SC Bus */
    /* These are accumulated cell voltages.  Individual converted
     * accumulated values are not necessary to keep, only to use
     * to obtain the individual cell voltages.
     */
    for (i = 0; i < NUM_BAT_CELLS; i++)
    {
        vbatta[i] = tlm.sensors.vbatta[i] * cnv_vbatts[0][i];
        vbattb[i] = tlm.sensors.vbattb[i] * cnv_vbatts[1][i];
    }
    tlm_cnv.vbatta = vbatta[TOP_CELL]; /* Battery Total Voltage is same as top cell voltage */
    tlm_cnv.vbattb = vbattb[TOP_CELL];

    /* These are individual cell voltages */
    tlm_cnv.vcellsa[0] = vbatta[0];
    tlm_cnv.vcellsb[0] = vbattb[0];
    for (i = 1; i < NUM_BAT_CELLS; i++)
    {
        tlm_cnv.vcellsa[i] = vbatta[i] - vbatta[i-1];
        tlm_cnv.vcellsb[i] = vbattb[i] - vbattb[i-1];

        tlm_cnv.vcellsa_avg += tlm_cnv.vcellsa[i];
        tlm_cnv.vcellsb_avg += tlm_cnv.vcellsb[i];
    }
    tlm_cnv.vcellsa_avg /= NUM_BAT_CELLS;
    tlm_cnv.vcellsb_avg /= NUM_BAT_CELLS;

    /* The spacecraft voltage */
    tlm_cnv.vscbus = tlm.sensors.vscbus * CNV_VSC;


    /* Convert Currents: Battery A, Battery B, SC Bus */
    for (ia = ib = isc = 0.0, i = 0; i < CURRENTS_SAMPLED; i++)
    {
        if (tlm.sensors.ibatta[i] & 0x8000)
            ia += -1.0*(((tlm.sensors.ibatta[i] & 0x0FFF)*CNV_IBATTS) - 5.0);
        else
```

281

```
                    ia += (tlm.sensors.ibatta[i] & 0x0FFF)*CNV_IBATTS - 5.0;

            if (tlm.sensors.ibattb[i] & 0x8000)
                ib += -1.0*(((tlm.sensors.ibattb[i] & 0x0FFF)*CNV_IBATTS) - 5.0);
            else
                ib += (tlm.sensors.ibattb[i] & 0x0FFF)*CNV_IBATTS - 5.0;

            isc += (tlm.sensors.iscbus[i] * CNV_ISC) - 5.0;
        }
        tlm_cnv.ibatta = ia/CURRENTS_SAMPLED;
        tlm_cnv.ibattb = ib/CURRENTS_SAMPLED;
        tlm_cnv.iscbus = isc/CURRENTS_SAMPLED;


        /* Call Battery Charge Monitory V,I, and Temperature data updater. */
/* Jah */
/*  bcm_tlm_update(); */

}  /* End of convert_ad() */


/*****************************************************************************
 *
 *   signed char  cnv_therm()
 *
 *   Convert A/D value for thermistor reading into a temperature (Celcius)
 *   using table look up for approximation.  A binary search is used to
 *   speed up the look up process.
 *
 *****************************************************************************/

int cnv_therm(unsigned int sample)
{
    register int mid, start, end;


    start = 0;
    end = THERM_TAB_SIZE-1;

    /* First see if the sample is less than the smallest table
     * value.  If so, this corresponds to a temperature greater
     * than that corresponding temperature.
     */
    if (sample < cnv_therm_tab[THERM_TAB_SIZE-1])
        return(THERM_HIGH + 1);

    /* Otherwise, the lookup will return the closet temperature,
     * including a temperature below the lowest in the table.
     */

    while ((end - start) > 1)
    {
        mid = (end + start)/2;

        if (sample == cnv_therm_tab[mid])
            break;
        else if (sample < cnv_therm_tab[mid])
            start = mid;
        else
            end = mid;
    }

    if ((end - start) > 1)
        return(THERM_LOW + mid);

    else
    {
        if ((cnv_therm_tab[start] - sample) <= (sample - cnv_therm_tab[end]))
            return(THERM_LOW + start);
        else
            return(THERM_LOW + end);
    }

}  /* End of cnv_therm() */


/*****************************************************************************
 *
 *   void check_tlm()
 *
 *   Check to see if the A/D acquisition has completed another sensor sweep,
 *   and if so convert the sensor data into a recent tlm record.  In addition,
 *   check to see if it is time to store the record to mass storage (Flash).
 *
 *****************************************************************************/
```

```
void check_tlm(void)
{
    static DWORD t= 0L;


    ad_check();


    if (samples_ready)
    {
        ad_collect();
        convert_ad();

        memcpy(&tlm_record.sensors, &tlm.sensors, sizeof(sensors_struct));

        tlm.etime = get_elapsed_time();
        tlm.tod = get_time();
        tlm_record.etime = tlm.etime;
        tlm_record.tod = tlm.tod;

        bcm_info(&tlm_record.bcm);

        if ((get_elapsed_time() - t) > TLM_RECORD_TIME)
            msu_save_tlm(tlm_record);
    }

}   /* End of check_tlm() */
```

End of tlm.h, tlm.c

APPENDIX K. TEST PLANS

This appendix contains the System Controller tests that were followed while evaluating the hardware and software.

Board Stuffing Tests (SC Hardware)

- Check distirbution of PCB logic power when board powered off.

- Check output voltage of DC-DC converter for regulated board power.

- Verfiy power on sensing circuitry.

- Check bus isolation buffers (54HC125).

- Verify clock from crystal oscillator.

- Check microprocessor Reset circuit.

- Verify microprocessor CLKOUT.

- Check power and ground to all ICs.

- Record current sinked to board.

- Verify CLKIN to all clocked ICs.

- Verify connectors.

- Verify signal filters to A/D inputs (RC-diode circuit).

Circuit Evaluation (SC Hardware and Software Device Drivers)

- Attach in-circuit emulation system.

- Perform and verify microprocessor Reset and microprocessor initialization.

- Test asynchronous mode of the SCC (use RS-232 terminal on other end).

- Check data and address buffers.

- Verify external interrupts are acknowledged by microprocessor.

- Test memory and chip selects for ROM, EDAC, and peripherals.

- Enable and verify EDAC via tests written for Oechsel [Ref. 21]. Include new test for EDAC reset and modified write back.

- Check power switching to Modem board (using the TPS2013).

- Test manual mode of the 82C55.

- Verify PCB write and read.

- Test manual mode of the LM12H458.

- Test conversion of IC temperature probe (LM50) into A/D.

- Enable SCC port A synchronous and check for Flag detection and creation.

Further Device Driver Tests (SC hardware and software)

- Verify Startup code: CPU init, peripheral init, memory check and clear, stack setup, floating point emulation init, passing control to main().

- Test clock generator.

- Test interrupt receive and acknowledge for multiple ISR operation.

- Check EDAC RAM wash chained into the clock generator.

- Verify SCC port B asynchronous modes (9.8, 19.2, and 38.4 kbits/sec).

- Check packet passing into Modem interface using SCC port B.

- Check terminal emulation and high-level print out display.

- Test STPI high-level command interface.

- Check PCB read and write routines (both non-interruptable and re-entrant routines).

- Test EPS control.

- Test TMUX channel select capability.

- Test A/D ISR for data acquisition.

- Test A/D data conversions.

- Verify Mass Storage SRAM and Flash read and write operations.

High-Level Software Tests (SC software)

- Check detection of STPI during bootup.

- Check telemetry saving and retrieving to and from the Mass Storage.

- Check Mass Storage Flash for recorded telemetry to allow for state preservation with system operations.

- Verify Battery Charge Monitor: ported from LabVIEW system. Single battery only (A, then B), dual battery (A and B), full charge, charge with solar simulation, maintaining battery charges, battery discharges, environmental tests, autonomous control for long periods (1 day, 3 day, 1 week).

- Check CRC generation and verification for saved telemetry to Mass Storage.

- Test RF control.

- Check software upload and transfer of control routines (allows new software to be loaded, including SCOS).

- Test interface for commands sent through the RF interface.

- Check stored telemetry download and erase.

- Verify scenario checking routines for anomolies.

LIST OF REFERENCES

1   "PANSAT Functional Requirements," SSD-S-SY-000, Space Systems Academic Group, Naval Postgraduate School, date Draft.

2   Johnson Space Center, NASA FAX, Subject: Space Shuttle Altitudes and Inclinations, April 1994.

3   Davinic, Nicholas M., "Evaluation Of The Thermal Control System Of The Petite Amateur Navy Satellite (PANSAT)," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1995.

4   Messenger, G.C. and Ash, M.S., *The Effects of Radiation on Electronics Systems*, Van Nostrand Reinhold, New York, NY, 1991.

5   Adams, Leonard, et al., "Proton Induced Upsets in the Low Altitude Polar Orbit, " *IEEE Transactions on Nuclear Science*, Vol. 36, No. 6, pp. 2339 - 2343, 1989.

6   Shimano, Y. T., et al., "The Measurement and Prediction of Proton Upset," *IEEE Transactions on Nuclear Science*, Vol. 36, No. 6, pp. 2344 - 2348, 1989.

7   "Space Product News, Special Edition, Integrated Circuits For The Space Environment," Harris Semiconductor BR-035, Harris Semiconductor, September 1993.

8   Kernigan, B. W., Ritchie, D. M., *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

9   Harbison, S.P., Steele, G. L., *C A Reference Manual*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

10   *CMOS Logic Selection Guide*, Harris Semiconductor, Melbourne, FL, 1994.

11   *Embedded Microcontrollers And Processors Volume II*, Order Number 270645, Intel Corporation, Mount Prospect, IL, 1992.

12   *80C186/C188 80C186XL/C188XL Users Manual*, Order Number 272164-001, Intel Corporation, Mount Prospect, IL, 1992.

13   "Embedded Intel 186 Microprocessor Family Frequently Asked Questions", http://developer.intel.com/design/faq/186faqs.htm, Intel Corporation, 1996.

14   "ICL8211, ICL8212 Programmable Voltage Detectors," Harris Semiconductor, AnswerFax Data Sheet 3184.1, 1992.

15   "Maxim 5V, Step-Down, Current-Mode PWM DC-DC Converters," Maxim 1995 New Release Data Book, Volume IV, Maxim Integrated Products, 1995.

16   "82C55A CMOS Programmable Peripheral Interface," Harris Semiconductor AnswerFax 2969, Harris Semiconductor, 1992.

17   "PA-100 Spread Spectrum Demodulator ASIC, Technical Data Sheet and User's Guide Rev.0," Paramax, Salt Lake City, UT, 1993.

18   "TPS 2010/TPS 2012/TPS 2013 Power Distribution Switches," Texas Instruments Data Sheet SLV097, Texas Instruments, 1994.

19   "M27C256 256k (32k x 8) CHMOS UV Erasable PROM," Intel Corporation Data Sheet 271008-067, 1992.

20   *Dataman S4 User Manual*, Dataman Programmers Ltd., Orlando, FL.

21   Oechsel, C. R, "Implementation Of Error Detection And Correction (EDAC) In The Static Random Access Memory (SRAM) Aboard Petite Amateur Navy Satellite (PANSAT)," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

22   "ASC630MS Radiation Hardened EDAC," Harris Semiconductor AnswerFax 2100, Harris Semiconductor, 1992.

23   "256k x 8 CMOS SRAM MSM8256-25/35/45/55," Mosaic Semiconductor Data Sheet Issue 2.2, Mosaic Semiconductor, 1993.

24   "LM12454/LM12H454/LM12458/LM12H458 12-Bit + Sign Data Acquisition System With Self-Calibration," National Semiconductor Data Sheet, National Semiconductor, 1995.

25   "LM12458 12-Bit Plus Sign Data Acquisition System with Self-Calibration", http://www.nsc.com/pf/LM/LM12458.html, National Semiconductor, 1996.

26   "DG411, DG412, DG413 Monolithic Quad SPST CMOS Analog Switches," Harris Semiconductor AnswerFax Data Sheet 5282.1, Harris Semiconductor, 1993.

27   "LM50B/LM50C SOT-23 Single-Supply Centrigrade Temperature Sensor," National Semiconductor Data Sheet, National Semiconductor, 1995.

28   *Am8530H/Am85C30 Serial Communication Controller Technical Manual*, Advanced Micro Devices, Sunnyvale, CA, 1992.

29   "Maxim ± 15 kV, ESD-Protected, +5V RS-232 Transceivers," Maxim 1995 New Release Data Book, Volume IV, Maxim Integrated Products, 1995.

30   Amateur Radio Relay League, *AX.25 Amateur Packet-Radio Link-Layer Protocol (Version 2.0)*, Washington D.C., 1984.

31   Price, H. E., *SCOS Reference Manual*, BekTek Corporation, Bethel Park, PA, 1992.

32   Price, H. E., *AX-25 Reference Manual*, BekTek Corporation, Bethel Park, PA, 1992.

33   Price, H. E., Ward, J., "PACSAT Protocol Suite," Proceedings Of The ARRL 9th Computer Network Conference, ARRL, Newington, CT, 1990.

34   *Writing ROMable Code in Microsoft C*, Systems And Software, Inc., Irvine, CA, 1990.

35   *Link And Locate++*, Systems And Software, Inc., Irvine, CA, 1990.

36   Lahti, C. A. , "The Design Of The Radio Frequency (RF) Subsystem Printed Circuit Boards For The Petite Amateur Navy Satellite (PANSAT)," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1996.

37   Horning, J., and Sheltry, J, "An Analog-to-Digital Acquisition System for PANSAT,", Space Systems Academic Group, 1993.

38   *The Temperature Handbook, Volume 28*, Omega Engineering, Stamford, CT, 1992.

39   *Flash Memory Products*, Advanced Micro Devices, Sunnyvale, CA, 1994.

40   McNamara, J. E., *Technical Aspects Of Data Communications*, Digital Press, Bedford, MA, 1982.

41   Strewinsky, F. H., "PANSAT Prototype Battery Test And Test Results," Research Report, Naval Postgraduate School, Monterey, CA, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
    8725 John J. Kingman Rd., Ste 0944
    Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library...2
    Naval Postgraduate School
    411 Dyer Rd.
    Monterey, California 93943-5101

3. Department Chairman, Code EC..1
    Department of Electrical and Computer Engineering
    Naval Postgraduate School
    Monterey, CA 93943-5121

4. Department Chairman, Code SP 1
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, CA 93943-5110

5. Dr. Rudolf Panholzer, Code Sp/Pz..1
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, CA 93943-5110

6. Randy L. Wight, Code Sp/Wt 1
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, CA 93943-5110

7. PANSAT Project Team, Code Sp 2
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, CA 93943-5110

8   Mr. and Mrs. Tony Horning...1
    25380 Vista del Piños
    Carmel, California 93923

9. Mr. James A. Horning, Code Sp/Jh..2
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, CA 93943-5110